

Building Maintainable Bots Using Code Analysis

Code Analysis => Maintainable, reliable, efficient, secure code.

It operates with a set of rules, each rule represents a coding style or convention => guidelines to analyze your code.

List of expectations/issues. The goal of code analysis is highlight problems that could exist in your code (coding errors, bugs, stylistic errors, omissions, security).

Catch potential issues early on and ensure that your code adheres to industry standards.

How can we make our automation code more uniform and comprehensible?

How can we reduce errors in our automation code?

What are coding best practices?

Code Analysis Key Features and Benefits:

- SCALING WITHOUT COMPROMISE => Establish best practices for building bots and ensures clear policies for enforcement.
- FREEDOM TO PICK AND CHOOSE THE BEST PRACTICES => Full control over which practices to enforce by selecting specific rules (ex.: Configure policies with variable name length rule and severity level as high or low).
- ASSESSING THE STATUS OF YOUR BOTS => Run the analysis from the Automation page and get summary reports across all bots in your directory.
- FIX THE IDENTIFIED ISSUES WITH THE ASSISTANCE OF THE BOT AGENT => Use the reports as guidance and make necessary code fixes right within the workbench.

User Roles:

- Lead RPA Developer => 1) Configure Code Analysis policies that define coding best practices in the organization; 2) Collaborate with CoE Management to monitor and report the status of code quality based on adherence to these policies; 3) Assist the developer community in producing higher-quality code.
- Citizen Developer => 1) Produce maintainable, reliable and secure automation by adhering to the policies defined by the Lead RPA Developer; 2) Find any bot violations and fix them by accessing reports; 3) Seek assistance from experienced developers.

Summary

In this video, we learned that:

- Code Analysis is a powerful tool for improving code quality and implementing best practices. It identifies coding errors, bugs, and stylistic issues, ensuring adherence to industry standards.
- Customers can define and enforce coding standards using Code Analysis. It enables scalability without compromising quality.
- Provides quick assessment of bot status and facilitates issue fixing.
- The Lead RPA Developer configures Code Analysis policies and monitors code quality.
- The Citizen Developer produces high-quality automation solutions by following the defined policies.

VIEW and MANAGE permissions => Log in as Control Room Administrator and configure Code Analysis policies.

Multiple named policies => flexibility to customize your policies based on your specific needs.

For each rule you can set severity level as high or low => this helps highlight criticality of the rule and prioritize your attention accordingly.

Then assign policies to automation folders

Administration

Audit log

Users

Roles

Licenses

Automation ANYWHERE Automation 360

Home

Automation

Activity

Manage

Scheduled

Event triggers

Devices

Queues

Global values

Credentials

OAuth connections

Packages

Administration

Audit log

Users

Roles

Licenses

Settings

Policies

Install licenses

View policies

Manage policies

View operations data

View operations details

Export operations details

Settings

> General

> Bots

> Policies

Code Analysis

When enabled this feature will enable all set Code Analysis policies and enable automated code analysis of bots in the repository.

☐ Enable ☒ Disable

> Packages

> Devices

> IQ Bot

> Credentials

> Bot agent bulk install

> Email

> Remote Git repository integration

> CoE Manager

> Login settings

Enabling Code Analysis means that the system will automatically scan all automation code files in the public repository that are located within folders assigned to a Code Analysis Policy

Administration

Audit log

Users

Roles

Licenses

Settings

Policies

Here you can configure multiple named policies for Code Analysis, each supporting different rules with specific settings and severity levels

You'll see a list of rules that can help improve your coding practices. For each rule, you can choose the severity level to indicate its criticality:

Create policy

Policy allows Automation Analyzer to enables rules that must be followed when building bots.

Policy details

Policy name Description (optional)

Task Bot rules

Use these rules to define best practices when building task bots.

General

Action Maximum ☐ Off The number of actions in the bot cannot exceed the specified maximum. ACT-GEN-001

Try-Catch Coverage ☐ Off All code must be within a Try Catch block. ACT-GEN-002

Infinite Loop Detection ☐ Off Mandates the check for infinite loops in the bot code. ACT-GEN-003

Variables

Variable Name Pattern ☐ Off Variable names must match specified naming pattern. VAR-NMG-001

Variable Name Length ☐ Off Variable names must be within specified minimum and maximum length. VAR-NMG-002

No Unused Variables ☐ Off Unused variable, all variables must be used within the bot. VAR-USG-001

Comments

Mandatory Header Comment ☐ Off The first action in the bot must be a Comment action. ACT-COM-001

Comment Coverage ☐ Off The number of Comment actions must be equal to or greater than the specified percentage of total command actions. ACT-COM-002

Hard coded values

No Hard Coded Delay ☐ Off The Delay action delay time must be set using a variable.

No Hard Coded File Path ☐ Off The File/Folder path must be set using a variable.

No Hard Coded Email Address ☐ Off The command action field must not contain a hard coded email address.

HIGH SEVERITY

- Mandatory for coding practices.
- Automations that violate this rule will not be deemed complete or acceptable for testing or production environments.

LOW SEVERITY

- Coding practice that you gradually want to implement over time.
- Practice that you want your developers to learn at present, but not obligatory for automation to be considered complete.

Code Analysis Enforcement => Lead RPA Developer (configure roles directly) and Control Room Administrator can collaborate to configure roles and violation thresholds.

Packages

Administration

Audit log

Users

Roles

Licenses

Manage connections

CODE ANALYSIS POLICY ENFORCEMENT

Enable enforcement for bot check-in

Allow check-in with low severity violations

Allow check-in with high severity violations

PACKAGE MANAGER

View packages

Create/Edit custom connector packages

Ex.: If Lead RPA Developer selects "Allow check-in with low-severity violation" then Bot Developer wouldn't be able to check-in bots with high-severity violations.

Summary

In this video, we learned that:

- Configuring Code Analysis involves ensuring View and Manage Policy permissions as the Lead RPA Developer.
- Multiple Code Analysis policies can be created with different rules and severity levels.
- Policies are assigned to automation folders for consistent enforcement.
- Severity levels (low or high) indicate the importance of adhering to coding best practices.
- Enforcement can be customized based on violation thresholds defined for roles by the Lead RPA Developer or Control Room Administrator.

How to run code analysis => view code analysis results => [automation page](#) or [bot editor](#)

Automation

Create new Import bots

Automation Private Bot Store

Private bots and files cannot be viewed by other people. If a bot or file has been checked out from the Public tab, it can be viewed and run by other people, but cannot be edited.

Folders

Bots

code_analysis

Document Workspace Pr...

IQ Bot Processes

My Tasks

Sample bots

Insights

TaskBotExample

Bot Store

Name Search

Files and folders (5)

Type	Name	Status	Source version	Code Analysis results	Size	Last modified	Modified by
Folder	bulk_checkin	N/A	N/A	N/A	N/A	3 minutes ago	bot_creator
Task Bot	Excel find text which is no...	New	N/A	Not scanned	36.59 KB	23 minutes ago	bot_creator
Task Bot	test_code_analysis	Checked out	3	High severity	2.75 KB	23 hours ago	bot_creator
Task Bot	test_migration	New	N/A	Low severity	1.27 KB	24 minutes ago	bot_creator
Task Bot	test1	New	N/A	Not scanned	2.32 KB	24 minutes ago	bot_creator

100 per page

If Code Analysis doesn't apply to the object because it's a folder or not a bot file, it will show as "N/A".

If Code Analysis hasn't been run on the bot, it will be displayed as "Not scanned".

When the bot has at least one high-severity violation, it will show as "High severity".

If the bot has at least one low-severity violation but no high-severity violations, it will display as "Low severity".

If there are no Code Analysis violations in the bot, it will show as "No violations".

<input type="checkbox"/>	Task...	confirmInvoiceDataBot	Public	6	N/A	High severity	247 KB	9	...
<input type="checkbox"/>	Form	confirmInvoiceDataE...	Public	3	N/A	N/A	15 Bytes	8	...
<input type="checkbox"/>	Task...	Display_Message	Public	2	N/A				...
<input type="checkbox"/>	Task...	displayMessageBot	Public	1	N/A	Not scanned	963 Bytes	1	...
<input type="checkbox"/>	Form	displayOrderForm	Public	2	N/A	N/A	644 Bytes	3	...

When you edit a bot, Code Analysis is automatically initiated in the background.

You can expand the code analysis option to locate the violations and find recommendation to fix them

Summary

In this video, we learned that:

- Running Code Analysis allows bot developers to evaluate bots for coding errors and adhere to predefined rules.
- Code Analysis results can be reviewed through the Automation Page or the Bot Assistant in the Bot Editor.
- By running Code Analysis, the status of each bot file is displayed, indicating whether there are high or low severity violations or no violations.
- Code Analysis can be initiated by running it directly from the Actions menu or viewing the bot history in the Bot Editor.
- Violations can be fixed by following the recommendations provided in the Code Analysis report and saving the bot to update the results.

Link to the course:

[Building Maintainable Bots Using Code Analysis - Running Code Analysis | Partner \(automationanywhere.com\)](https://www.automationanywhere.com/partner/building-maintainable-bots-using-code-analysis-running-code-analysis/)

Building Action Packages for Automation 360

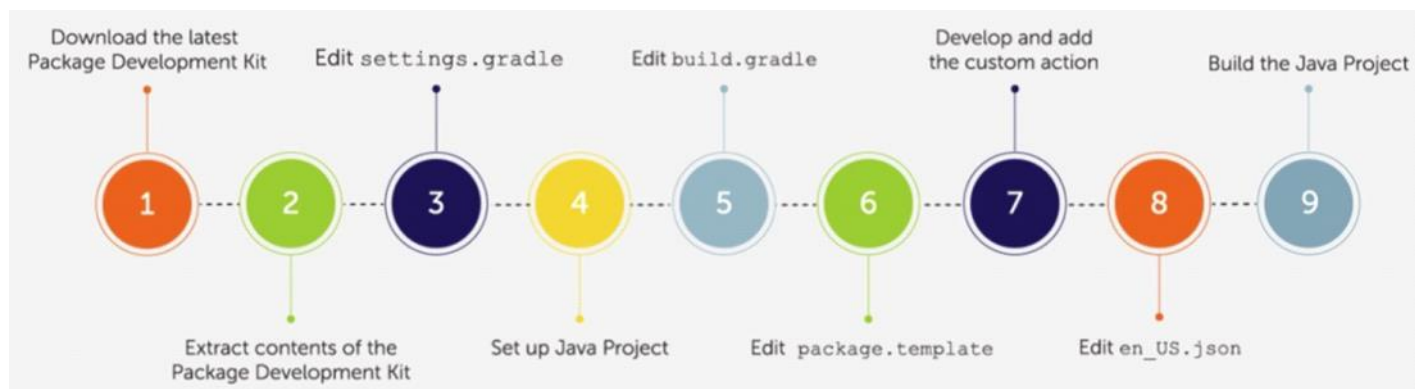
Packages are Java Archive (or JAR) files that contain applications used to create bots.
Automation 360: modular architecture => decoupling actions from the main product.
Useful because you can update packages without having to re-install Automation 360 (they can be independently updated) > Multiple version of the same package can coexist in an environment.

Automation 360 Package Software Development Kit > can be used to build custom actions.

Updating the package does not update your existing bots. You should manually select the latest version of the package to be used in the existing bots.

Package can be managed by Role Based Access Control (RBAC).
If a package is currently being used by a bot, it cannot be deleted.

Audit log to troubleshooting when you need to trace all package related changes (Add, Delete, Disable).



Annotation are used to provide supplement information about a program and always preceded with '@'.

[Annotations \(automationanywhere.com\)](https://automationanywhere.com)

Annotation	Additional Note/Condition
BotCommand Makes the type eligible to be treated as an action.	You can define 3 types of actions commandType property. <ul style="list-style-type: none">• Command• Condition• Iterator Note: In this course we are using only the Command action.
CommandPkg Makes the type eligible for creation of action package.json .	This annotation must be used with BotCommand to take effect. Pkg would participate in the activity only when this annotation is present.
Execute Method annotated with this annotation will participate in the execution of BotCommand . This annotation is the entry point to the action function from the A2019 bot run environment.	Exactly one method needs to be annotated when BotCommand annotation is present on the type. Failure to do so will result in compilation error.
Idx Makes the annotated element part of the hierarchy utilized for code and resource generation.	Without this annotation no BotCommand related element annotations is processed.
Pkg Makes an element participate in creation of package.json .	This annotation is ignored when Idx is not present.

Void > use this return type if your action does not return any value.

To support labels and descriptions in other languages, additional json files are provided. Edit the respective file and provide labels and descriptions in the required language.

Deploying Action Packages => Upload Package Permission

Intro to Packages + Machine Setup

[Automation Anywhere Automation 360: Micah Smith's Intro To Packages \(youtube.com\)](#)

What is a Package? A **Package** is a **collections of configurable actions that can be used as the building blocks of bots**.

Automation 360 provides for several methods of filling your development environment with packages: 1) **Out of Box** (70+ packs); 2) **Bot Store** (45+ free packs + Source Code available on GitHub, es: [AutomationAnywhere/A360-FileConversionPackage: Custom Automation 360 package for performing various file conversions \(github.com\)](#)); 3) **Custom Development** > the package SDK includes samples of creating each of the "action" types.

When you install those custome packages either from Bot Store or from your custom development, they integrate perfectly with the existing product > it'll sit right next to the out of box packages and you can add your icons and add your logos and your descriptions > all of that can look exactly like the core product.

Bots in Automation 360 are a listing of actions and their corresponding configurations.

RUNTIME > When tasks are sent to a Bot Runner, the bot and all referenced packages are sent to the runner for processing:

- If the package versions already exist locally, nothing is downloaded.
- If the package version doesn't locally exist, it's downloaded and stored.

[Automation Anywhere Automation 360: Micah Smith's Intro To Package Development: Machine Setup \(youtube.com\)](#)

Machine Setup

- JDK 11 ([JDK 11 Documentation - Home \(oracle.com\)](#))
- Java IDE (IntelliJ)
- Gradle plug-in v5.*.* in the IDE (Included out of box in IntelliJ)
- Installed Bot Agent
- Package SDK

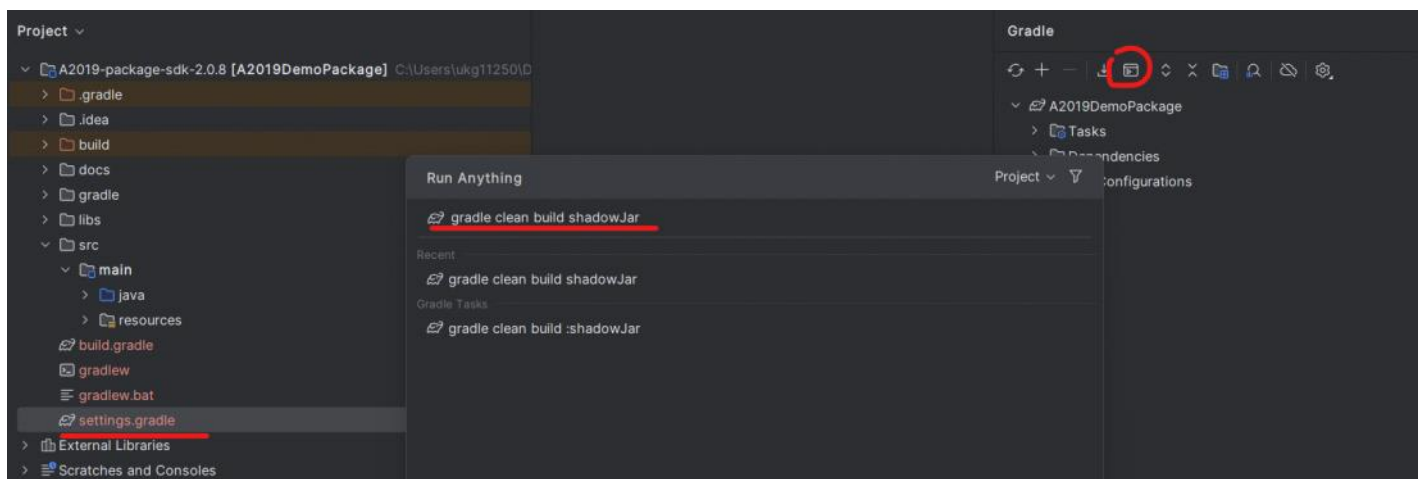
[Set up the Java project \(automationanywhere.com\)](#)

[Develop a sample package \(automationanywhere.com\)](#)

If you're using an older version of AA control room, just make sure that you're using the appropriate SDK version that matches up with your version of the control room.

Ex.: you might have issues if you try to build it on .19 (SDK) but then install it and run it on a .17 environment (CR).

=> If you had a .17 package (SDK) you should install fine on a .19 environment (CR).



Build Your First Automation Package

[#AAillustrates: Build Your First Automation 360 Package with Micah | Automation Anywhere \(youtube.com\)](#)

Concepts Covered

- Building
- Testing
- Deploying
- Validating a Package

Package > Actions (Commands), Conditional Statements, Iterators, Triggers, Variables.

SET UP THE REQUIRED DEVELOPMENT ENVIRONMENT

Set up an Integrated Development Environment (IDE) for Java.

Prerequisites:

- Java IDE > Eclipse or IntelliJ ([Download IntelliJ IDEA – The Leading Java and Kotlin IDE \(jetbrains.com\)](#))
- Java Development Kit (JDK 11)
- Latest Gradle plug-in in the IDE
 - o [Gradle in IDEs](#)
 - o [Gradle User Manual](#)
 - o [Gradle | IntelliJ IDEA Documentation \(jetbrains.com\)](#)
- Automation 360 SDK ([Previous Package SDK Release Notes \(automationanywhere.com\)](#))

USE CASE

Create a custom package with an action that will take a file path and return the size of a file in bytes.

Steps:

- Start with the Automation 360 SDK
- Delete all sample classes except the Concatenate class
- Create a GetFileSize class
- Create a test class to validate our code
- Build and test the Jar created from our project in a bot

Package Level: Attributes

Intro to Package Development

Attribute	Where it Appears
Settings.gradle: rootProjectName	only shows up in your Java IDE and is used for the name of the jar file created on build.
Package.template: name	Name of the package – only shows on the install page and package detail page
Package.template: label	Label for how your package will be named/displayed in the package list and dev interface - uses locales json by default.
Package.template: description	Description for your package available in the package details and on mouse over of the package in the development interface

Package Details

Name: ActiveDirectory

Vendor: Automation Anywhere

Description: Provides actions to perform Active Directory operations.

Package.template: name

Package.template: description

Package.template: label: Active Directory

Provides actions to perform Active Directory operations.


```
settings.gradle (A2019DemoPackage) x
1  /*
2   * This file was generated by the Gradle 'init'
3   *
4   * The settings file is used to specify which pr
5   *
6   * Detailed information about configuring a mult
7   * in the user manual at https://docs.gradle.org
8   */
9
10 rootProject.name = 'A360FileDetails'
11
```

Project ▾

- gradle
- libs
- src
 - main
 - java
 - com.automationanywhere.botcommand.samples
 - commands
 - basic
 - desktopoperation
 - types
 - Concatenate
 - ui
 - conditional
 - iterator
 - trigger
 - variable
 - resources
 - com.automationanywhere.botcommand.samples
 - icons
 - locales
 - package.template

settings.gradle (A2019DemoPackage) package.template x en_US.json

```
1 {
2   "name": "FileDetails",
3   "label": "[[label]]",
4   "description": "[[description]]",
5   "group": "",
6   "artifactName": "",
7   "packageVersion": "",
8   "codeVersion": "",
9   "author": "",
10  "commands": []
11 }
```

Project ▾

- desktopoperation
- types
 - Concatenate
- ui
- conditional
- iterator
- trigger
- variable
- resources
 - com.automationanywhere.botcommand.samples
 - icons
 - locales
 - de_DE.json
 - en_US.json
 - es_XL.json
 - fr_FR.json
 - it_IT.json
 - ja_JP.json
 - ko_KR.json
 - pt_BR.json
 - ru_RU.json
 - zh_CN.json
 - zh_TW.json
 - package.template

settings.gradle (A2019DemoPackage) package.template en_US.json x

```
1 {
2   "label": "File Details",
3   "description": "Return basic file details",
4   "Concatenate.label": "Concatenate",
5   "Concatenate.description": "Concatenates two strings",
6   "Concatenate.node_label": "{{firstString}} with {{secondString}}",
7   "Concatenate.return_label": "Assign the output to variable",
8   "Concatenate.firstString.label": "First string",
9   "Concatenate.secondString.label": "Second string",
10  "Uppercase.label": "Uppercase",
11  "Uppercase.description": "Uppercase",
12  "Uppercase.node_label": "{{caseType}} of {{sourceString}}",
13  "Uppercase.return_label": "Assign the output to variable",
14  "Uppercase.sourceString.label": "Source string",
15  "Uppercase.caseType.label": "Provide which part of string to upp",
16  "Uppercase.caseType.2.1.label": "Make whole string uppercase",
17  "Uppercase.caseType.2.1.node_label": "all",
18  "Uppercase.caseType.2.2.label": "Make only first character of th",
19  "Uppercase.caseType.2.2.node_label": "first character",
20  "CredentialTypeDemo.label": "Credential Demo",
21  "CredentialTypeDemo.description": "Sample action showing how cre",
22  "CredentialTypeDemo.credentials.label": "Provide the credentials",
23  "SessionDemo.label": "Sessions Demo",
24  "SessionDemo.description": "Sample action showing how sessions a",
25  "SessionDemo.name.label": "Please provide the session name",
26  "TextTypeDemo.label": "Text Demo",
27  "TextTypeDemo.description": "Sample action showing how to displ
```


Action Level: Annotation & Attributes

Action:

- @BotCommand > makes a class eligible as an action
- @CommandPkg > establishes GUI label elements
- @Execute > entry point for action operation

Within an action we can have multiple input fields.

Input Field (annotations):

- @Idx > index position of the field (each must be unique; i.e. different index position for field 1, field 2, etc..)
- @Pkg > UI labels including field title and description (once I've selected an action on the right hand side and I can set all the different configuration for that particular action, that's where these labels and descriptions are showing up)
- @NotEmpty > (optional) for requiring a field value is set

@CommandPkg Labels

Intro to Package Development

Attribute	Where it Appears
Name	Internal name for action – must be unique within package
label	Name of the action as appears in package details and bot building screens.
node_label	Sub-label that shows under the action in bot building screens
description	Description of action that shows on hover in bot building screens
return_label	Label of the returned value field
return_description	Optional description of the returned value
icon	Name of svg to be used as icon in displaying on bot building screens



@Pkg Label & Description

Intro to Package Development

- Field-level Label & Description

Attribute	Where it Appears
@Pkg: label	Label of the field which appears above the input in bot building screens
@Pkg: description	Sub-label that shows under the input in bot building screens (here used for clarification on field formatting)

Package.template: label : @CommandPkg: label



Recap

Intro to Package Development

1. Starting with the SDK commands provides for the “bones” for custom development
2. Knowing where different labels/names show up enables custom packages that look and feel just like those provided out of box
3. Test classes let you test code before actually building/deploying into the Control Room

Useful links:

[Tutorial: Building An Automation 360 Package | Community \(automationanywhere.com\)](#)

[Creation and function annotations \(automationanywhere.com\)](#)

[Package SDK examples \(automationanywhere.com\)](#)

[Create custom variables using Package SDK \(automationanywhere.com\)](#)

About Gradle

[Gradle tutorial for complete beginners \(youtube.com\)](https://www.youtube.com/watch?v=8j338D0U8j4)

Gradle = Build Automation Tool > Primarily used for building Java Applications



COMPILATION = Process of generating the bytecode => Transform your application from human readable code to machine readable code (bytecode).

Bytecode can be run on the Java virtual machine (JVM).
You can compile without build tool.

JDK (Java Development Kit) comes with the Java compiler command called `javac.exe` > You call `javac` with a list of all the `.java` source files.

Jar files > functionality provided by third parties (ex.: Spring Boot, Apache, Guava).

```
javac -cp lib1.jar;lib2.jar;lib3.jar MyProgram.java
```

Compiling java code manually gets really tedious.

Main reason build tools exist > to make compilation a simple error-free and repeatable process.

Packaging your application means putting it into a format that can easily be published and deployed.
For Java applications => `.jar`, `.war` or `docker img`

```
$jar -tf gradle-tutorial.jar
META-INF/
META-INF/MANIFEST.MF
com/
com/tomgregory/
com/tomgregory/GradleTutorial.class
$
```

Gradle is a build tool designed specifically to meet the requirements of building java applications (compiling, testing and packaging your application).

```
c:\workspace\gradle-tutorial>gradlew build

BUILD SUCCESSFUL in 2s
4 actionable tasks: 4 executed
c:\workspace\gradle-tutorial>
```

How does Gradle know how to build your application?

- On a high level you have to describe what type of application you're trying to build (i.e. Java).
- You need to tell about any libraries your application depends on (Dependencies)
- Any other configuration specific to your application such as special compiler or testing options.

In gradle, you provide this configuration in a file called the **build script (build.gradle)**

Gradle is designed to perform extremely quickly.

INCREMENTAL BUILD => once you've compiled your application once, when you try to compile again after not having changed anything, gradle knows it doesn't need to recompile.

Gradle has a very advanced dependency management system: at a basic level, when you define dependencies and run your build, gradle downloads them automatically from the internet => managing and updating dependencies simple.

Maven vs. Gradle => Gradle made defining your build a lot less verbose, because it used a code-based build script, rather than xml build file. Since Gradle took a code-first approach, writing custom plugins and reusing build logic became a lot easier.

The build script is written as code and this can be either Groovy or Kotlin.

The **project** is the highest-level construct representing the application you want to build, including the configuration of how to build it (container for everything that gradle knows about your application). So if you've got an application's source code sitting in a repository, an accompanying Gradle project also gets committed into the repository with all the information needed to build the application.

- **settings.gradle** sets up some high level configuration for the project (ex.: Project name => `rootProject.name = 'projectName'`, otherwise gradle will by default use the directory name).
- **build.gradle** is the build script configuration file, describing your application to Gradle so it can build it (ex.: Java application with a particular set of dependencies).
- **gradlew** and **gradlew.bat** are known as the gradle wrapper scripts: gradlew is for Linux and Mac environments and gradlew.bat is for Windows > these let you build an application without having to download and install Gradle. When the wrapper is executed, it will automatically download Gradle and cache it locally. Normally you always build your application with the wrapper as it ensures it gets built with the correct version of Gradle.
- **.gitignore** > it configures git so that the .gradle and build directories aren't committed into version control, everything else gets committed though.
- **.gradle directory** is a local cache managed by Gradle.
- **build directory** is where Gradle creates any build outputs such as the compiled Java code or Jar files.

Each Gradle project can have a Build script. Gradle tasks are individual build actions you can run from the command line. You might have a task to compile your Java code, a task to test the code, a task to package the compiled classes into a Jar file. To run a task: *gradlew <task-name>*.

```

Build Setup tasks
-----
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Help tasks
-----
buildEnvironment - Displays all buildscript dependencies declared in root project 'gradle-tutorial'.
dependencies - Displays all dependencies declared in root project 'gradle-tutorial'.
dependencyInsight - Displays the insight into a specific dependency in root project 'gradle-tutorial'.
.
help - Displays a help message.
javaToolchains - Displays the detected java toolchains.
outgoingVariants - Displays the outgoing variants of root project 'gradle-tutorial'.
projects - Displays the sub-projects of root project 'gradle-tutorial'.
properties - Displays the properties of root project 'gradle-tutorial'.
tasks - Displays the tasks runnable from root project 'gradle-tutorial'.

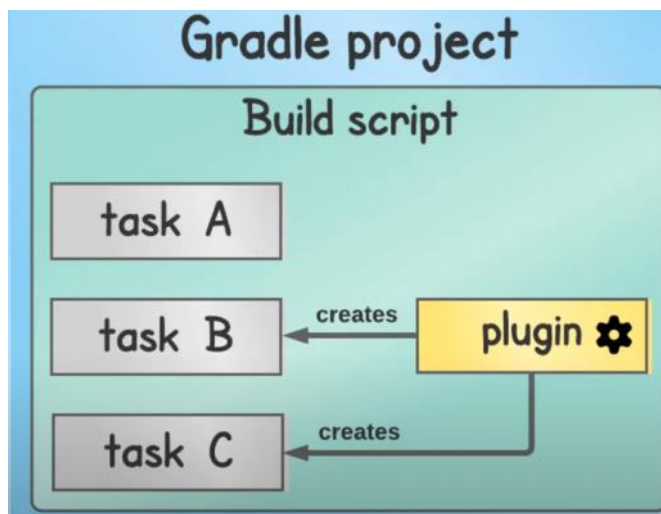
To see all tasks and more detail, run gradlew tasks --all

To see more detail about a task, run gradlew help --task <task>

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
c:\workspace\gradle-tutorial>

```

Gradle plugin > when you apply a plugin in your build script it automatically adds tasks to your project which you can run to achieve some particular outcome. For example, the Java plugin automatically adds tasks to compile, test and package your application. Using plugins means you don't have to reinvent the wheel



Groovy essentials

Groovy is a language which, like Java, runs on the Java Virtual Machine (JVM). It was chosen as a language for Gradle build scripts because of its dynamic nature, which allows your build to be concisely configured using what's called the **Gradle Groovy DSL (Domain Specific Language)** => writing Gradle build scripts involves writing Groovy code but doing it in a way that uses the Gradle APIs.

Groovy is a scripting language, so you can write code outside of a class and execute it.

Semicolons at the end of a line are not required.

Brackets are optional when passing parameters to a build if the method has at least one parameter.

```
Groovy code
1 def myVar = 'Executing as a script'
2 println myVar

Execution result
Executing as a script
```

```
1 def multiply(first, second) {
2     println first * second
3 }
4 multiply 2, 3

Execution result
6
```

no brackets
(a.k.a. parantheses)

You can define closures using curly brackets > Closures are blocks of code that get passed around and executed at a later point.

```
1 def myClosure = {
2     println 'Executing closure'
3 }
4 myClosure()
```

Execution result
Executing closure

You don't need to know much of Groovy to work in Gradle or build scripts, because the Groovy DSL uses only a subset of the Groovy language features

Building a Java Application

By default Gradle expects Java classes to live in **src/main/java**.

In **build.gradle** > the way we apply a plugin is to call the **plugins** method and pass a closure as argument.

```
GradleTutorial.java x build.gradle x
1 plugins {
2     id 'java'
3 }
```

can use single quotes in Groovy

What tasks has the Java plugin added? The command **"gradlew tasks"** prints out all the tasks.


```

Build tasks
-----
assemble - Assembles the outputs of this project.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
classes - Assembles main classes.
clean - Deletes the build directory.
jar - Assembles a jar archive containing the main classes.
testClasses - Assembles test classes.

Build Setup tasks
-----
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Documentation tasks
-----
javadoc - Generates Javadoc API documentation for the main source code.

Help tasks
-----
buildEnvironment - Displays all buildscript dependencies declared in root project 'gradle-tutorial'.
dependencies - Displays all dependencies declared in root project 'gradle-tutorial'.
dependencyInsight - Displays the insight into a specific dependency in root project 'gradle-tutorial'.
help - Displays a help message.
javaToolchains - Displays the detected java toolchains.
outgoingVariants - Displays the outgoing variants of root project 'gradle-tutorial'.
projects - Displays the sub-projects of root project 'gradle-tutorial'.
properties - Displays the properties of root project 'gradle-tutorial'.
tasks - Displays the tasks runnable from root project 'gradle-tutorial'.

Verification tasks
-----
check - Runs all checks.

```

Let's focus on Build task which "Assembles and tests this project".

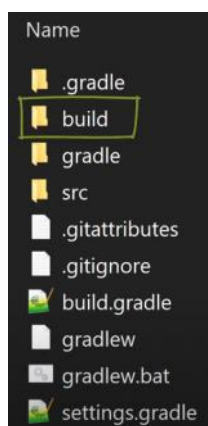
```

c:\workspace\gradle-tutorial>gradlew build

BUILD SUCCESSFUL in 2s
2 actionable tasks: 2 executed
c:\workspace\gradle-tutorial>

```

We've got a new directory called "build"



The way we add the main class manifest attribute is by configuring the Jar task. To configure it, we call a method Jar, passing a closure, then manifest passing enclosures and attributes passing it a map. The values of this map are the additional manifest attributes, in this case it's a key of main class and a value of the fully qualified class name.

```

GradleTutorial.java  build.gradle
1 plugins {
2     id 'java'
3 }
4
5 jar {
6     manifest {
7         attributes 'Main-Class': 'com.tomgregory.GradleTutorial'
8     }
9 }

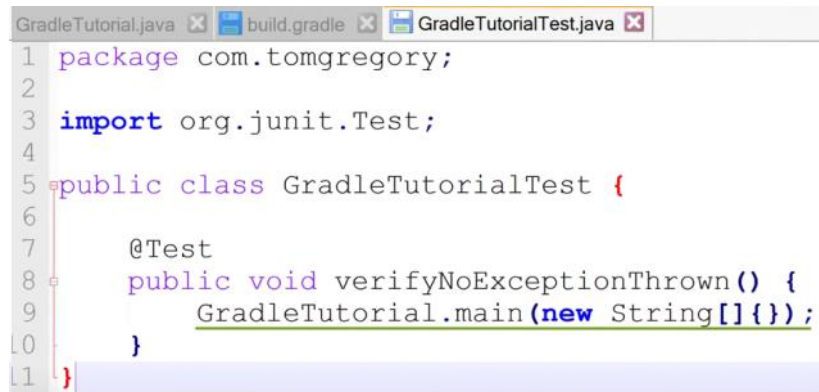
```

We should be able to execute it now with the Java command.

```
c:\workspace\gradle-tutorial>gradlew build  
BUILD SUCCESSFUL in 1s  
2 actionable tasks: 1 executed, 1 up-to-date  
c:\workspace\gradle-tutorial>java -jar build\libs\gradle-tutorial.jar  
Gradle 4tw!
```

Testing

In Gradle all tests go into **src/test/java**.



```
GradleTutorial.java x build.gradle x GradleTutorialTest.java x  
1 package com.tomgregory;  
2  
3 import org.junit.Test;  
4  
5 public class GradleTutorialTest {  
6  
7     @Test  
8     public void verifyNoExceptionThrown() {  
9         GradleTutorial.main(new String[]{});  
10    }  
11 }
```

Adding dependencies and repositories

We specify dependencies in our build.gradle script by calling the dependencies method passing a closure, within that we call the testImplementation method passing it a map of the dependencies (group, name and version). Any dependencies you pass to testImplementation will end up on the test compile and runtime class paths.

This means we'll be able to compile and run the test which has a reference to the JUnit 4 library



```
GradleTutorial.java x build.gradle x GradleTutorialTest.java x  
1 plugins {  
2     id 'java'  
3 }  
4  
5 jar {  
6     manifest {  
7         attributes 'Main-Class': 'com.tomgregory.GradleTutorial'  
8     }  
9 }  
10  
11 dependencies {  
12     testImplementation group: 'junit', name: 'junit', version: '4.13.2'  
13 }
```

We need to tell which repository to pull JUnit 4 from, which is the maven central repository.

We do that in the build script by calling the repositories method passing a closure and then calling the mavenCentral() method. We have to use brackets in this case to call mavenCentral() because in Groovy you can only leave out the brackets if the method has one or more parameters

mvnrepository.com/artifact/junit/junit/4.13.2

MVNREPOSITORY Search for groups, artifacts, categories Search

Home » junit » junit » 4.13.2

JUnit » 4.13.2

JUnit is a unit testing framework for Java, created by Erich Gamma and Kent Beck.

License	EPL 1.0
Categories	Testing Frameworks
Organization	JUnit
HomePage	http://junit.org
Date	(Feb 13, 2021)
Files	jar (375 KB) View All
Repositories	Central

Indexed Artifacts (23.8M)

Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries

```

1 plugins {
2     id 'java'
3 }
4
5 jar {
6     manifest {
7         attributes 'Main-Class': 'com.tomgregory.GradleTutorial'
8     }
9 }
10
11 repositories {
12     mavenCentral()
13 }
14
15 dependencies {
16     testImplementation group: 'junit', name: 'junit', version: '4.13.2'
17 }

```

Useful links:

[Gradle | Installation](#)

[tkgregory/gradle-tutorial: Gradle tutorial for complete beginners \(github.com\)](#)

Integrating Third-Part Applications and Automation 360 using Control Room APIs

Application Programming Interface (API) allows a custom application to interact with your software and exchange data.

Ex.: Airlines applications exposes certain end points => travel aggregator application can interact with it using APIs.

[Control Room APIs \(automationanywhere.com\)](https://automationanywhere.com)

- Authentication API
- User Management API
- Audit API
- Device API
- Automation API
- Credential Vault API
- Bot Execution Orchestrator API
- Bot Insight API
- Repository Management API
- BLM API
- Device Pool API
- License API
- Bot Deploy API
- Workload Management API
- Migration API (Data Migration from 10x)

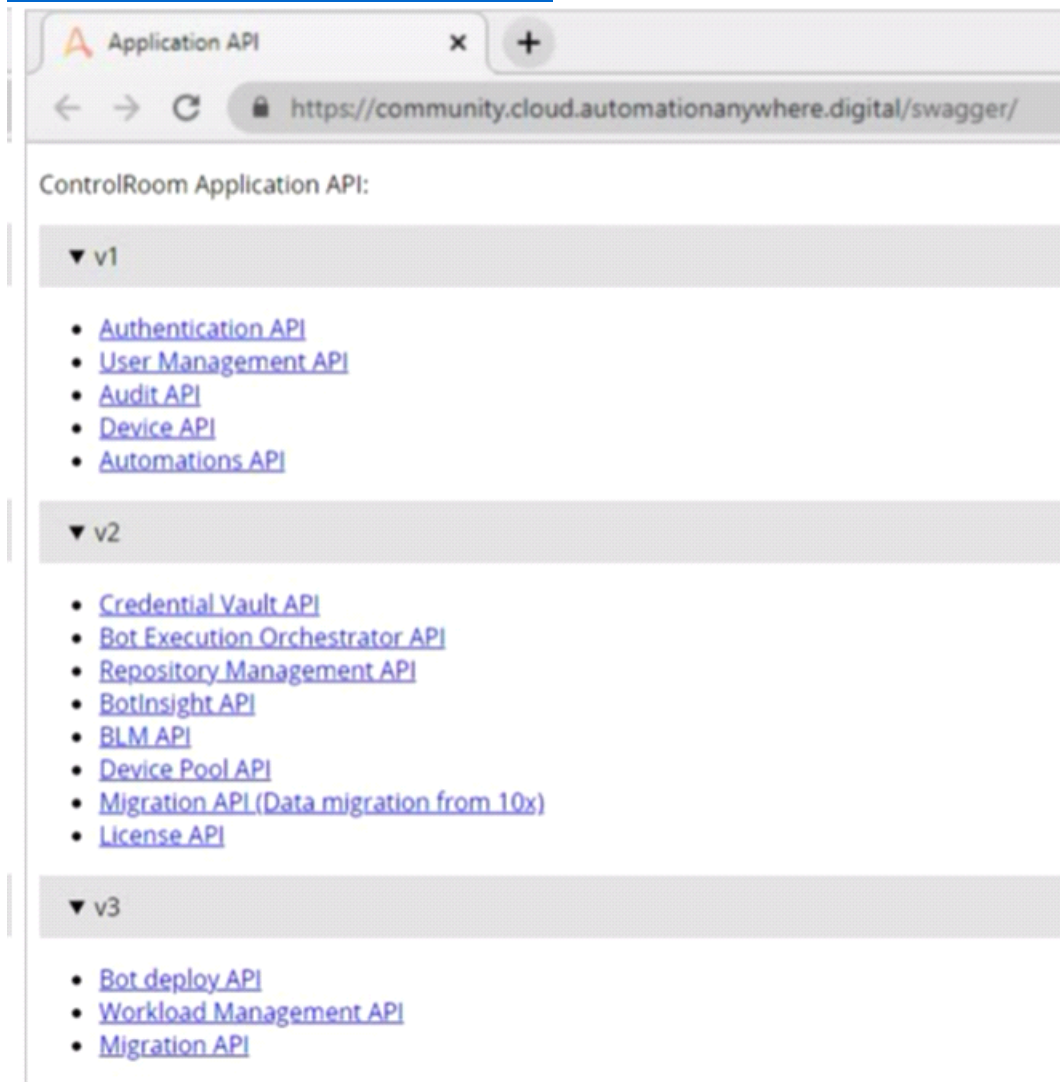
Automation 360 Control Room APIs enable the third-party applications to consume RPA, orchestrate bots and manage the RPA data based on events; enhance safety with token-based access; Use encrypted communication through TLS 1.2 enabled HTTPS channel.

Each Control Room API has a structure that define how the API is used and how the results are communicated.

Component	Example
Base URL	https://<controlroomURL>/v1
Endpoint	/authentication
Method	GET or POST
Header	-H X-Authorization: <token>
Body	<pre>{ "username": "string", "password": "string", "apiKey": "string" }</pre>
Response Code	200 OK
Response Body	<result of the API call>

HTTP Status Code	Description
200	OK
400	Bad request
401	Authentication required
403	Unauthorized access
404	Resource not found
407	Authentication required
409	Conflict
500	Internal server error

[Application API \(automationanywhere.digital\)](https://community.cloud.automationanywhere.digital/)



The request body can include following options for the API:

- Filter options
- Pagination options
- Input parameters

Each API request sent to Automation 360 Control Room needs to be accompanied by an Access token, that is used to authorize the API call.

You can use the Authentication API to generate, refresh and manage JSON Web Token (JWT) that are require for authorization in all Automation 360 Control Room APIs => use a token instead of a user's password (useful for organizations with SSO).

Use POST method to generate JWT. Input parameters for Authentication API:

- Username (AA user).
- Password (AA user password).
- apiKey > used in place of a psw for users that are assigned to the API key generation role.
Generated via My Settings in user's Control Room.

POSTMAN > Once you launch Postman, you can set up environments where you can list the details about the target server you will be accessing using APIs.

User Management API enables you to create, search, update or delete roles and users in your Automation 360 control room.

Role APIs


POST	/roles/list	Search for roles	🔒
POST	/roles	Create role	🔒
GET	/roles/{id}	Get role by ID	🔒
PUT	/roles/{id}	Update role	🔒
DELETE	/roles/{id}	Delete role	🔒

- Searches for roles
- Creates new roles
- Retrieves the details of a specific role based on a unique role ID
- Modifies an existing role based on a unique role ID
- Deletes an existing role based on a unique role ID

User APIs

POST	/users	Create user	🔒
POST	/users/list	Search for users	🔒
GET	/users/{uid}	Get user details	🔒
PUT	/users/{uid}	Update user details	🔒
DELETE	/users/{uid}	Delete user	🔒


- Creates new user
- Searches for all or a subset of users in the Automation 360 Control Room
- Retrieves user details based on a unique user ID
- Modifies an existing user based on a unique user ID
- Deletes an existing user based on a unique user ID

Request URL	https://<your_control_room_url>/v1/usermanagement/users/list	
Method	POST	
Body	Filter Results	Pagination
	<pre>{ "filter": { "operator": "NONE", "operands": [null], "field": "string", "value": "string" } }</pre>	<pre>{ "page": { "offset": 0, "length": 0 } }</pre>
<div><div></div><div>User making the API call should have:<ul style="list-style-type: none">• Successfully obtained an access token• View and manage users and roles permission</div></div>		

Request URL	http://<your_control_room_url>/v1/usermanagement/			
Method	POST	POST	PUT	DELETE
	Retrieve role info	Create Role	Update Role	Delete Role
Body	<pre>{ "id": 0, "name": "string", "description": "string", "permissions": [{}], "principals": [{}] }</pre>			

- v1 - Device API > Provide a list of Bot Runners users

- v2 - Bot Execution Orchestrator API > Monitors bot progress through a set of Automation 360 Control Room APIs
- v3 - Bot Deploy API > Deploys the bots in Automation 360 Control Room

POST	/runasusers/list	Fetch the results of the available run as users
repository		
POST	/file/list	Searches files in repository
DevicePool		
POST	/pools/list	Returns list of device pools available to the user
POST	/automations/deploy	This API is to deploy a bot using runAs Users. 

Useful links:

[Send API requests and get response data in Postman | Postman Learning Center](#)
[Navigating Postman | Postman Learning Center](#)

Introduction to the Control Room API

[Introduction to the Control Room API | Learn RPA | Micah Smith Automation Anywhere \(youtube.com\)](#)

What is an API?

An API is an Application Programming Interface that receives requests, does some actions and sends a response.

=> That action could be fetching data to send in the response, performing calculations on the request data, etc.

Ex.: You log in to LinkedIn.com. Your browser (the client) sends a request to LinkedIn's API. Your browser receives the response from the API, interprets the response and display the results as a webpage.

In AA, 2 types of APIs:

- 1) The kind of rest web service action package that's available and you can use that to call out to other APIs.
- 2) Control room APIs > a series of endpoints that are available to you as a developer to be able to consume on your own control room.

The AA CR provides various public APIs which allow you to control/manage your AA environment. This API includes most of the same functions you can do from the CR itself > User management, migrating bots, repository management, bot deployment..

[Control Room APIs \(automationanywhere.com\)](#)

APIs allow applications to expose certain functionality to developers for consumption

[How to Trigger a Bot from the Control Room API | #AAillustrates with Micah Smith Episode 15 \(youtube.com\)](#)

Setting Up Package Access Using Role-based Access Control

All the packages in Automation 360 are available to all the users to build a bot, by default. This is also true for custom packages that you create using the Automation 360 Package Development Kit. As a Control Room Administrator, you can now control access to these packages to make selected packages available to only a selected group of users.

This is especially useful if you want to restrict some packages to pro developers. For example, you do not want all the bot developers to use packages such as Python script, VB Script, and JavaScript. In such cases, **the Control Room Administrator can restrict access to the package so that only specified users can access it.**

The benefits of Role-Based Access Control for Packages include:

- Enterprise-wide access policy: Group or role-based package access enables you to establish clear policies restricting package access to all users or users with specific roles.
- Centralized policy enforcement: Group or role-based access to automation provides enhanced security, confidence, and ease of use for developers.
- Access based on business needs: Based on the business need, you can create the required groups and expose the appropriate library of packages to these groups. You can also manage the phased rollout of package access.
- Simplified automation for Citizen developers: With group or Role-Based Access Control, you can hide complex packages from the Action palette for Citizen developers so that they can focus on easy-to-complete automation.
- Automation reuse with compliance: Pro developers can create utility bots using complex packages and make them available to Citizen developers to use as cloned bots in their private workspace or inside public bots.

Enterprise-wide access policy

Group or role-based package access enables you to establish clear policies restricting package access to all users or users with specific roles.

Centralized policy enforcement

Group or role-based access to automation provides enhanced security, confidence, and ease of use for developers.

Access based on business needs < >

Based on the business need, you can create the required groups and expose the appropriate library of packages to these groups. You can also manage the phased rollout of package access.

Simplified automation for Citizen developers < >

With group or Role-Based Access Control, you can hide complex packages from the Action palette for Citizen developers so that they can focus on easy-to-complete automation.

Automation reuse with compliance < >

Pro developers can create utility bots using complex packages and make them available to Citizen developers to use as cloned bots in their private workspace or inside public bots.

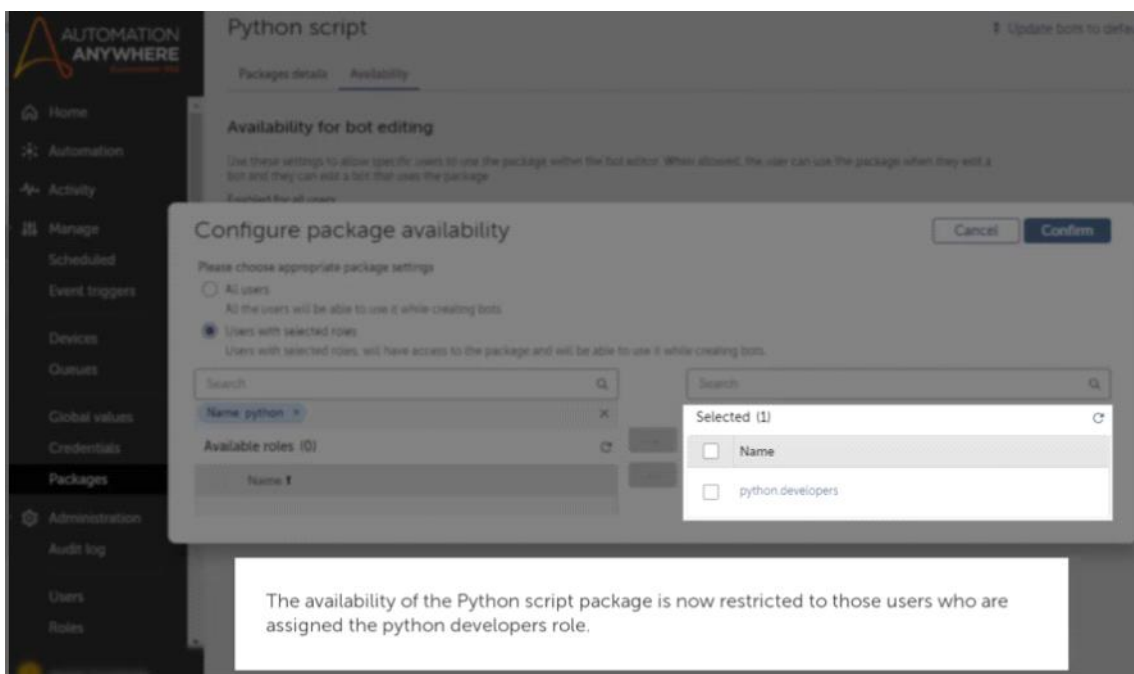
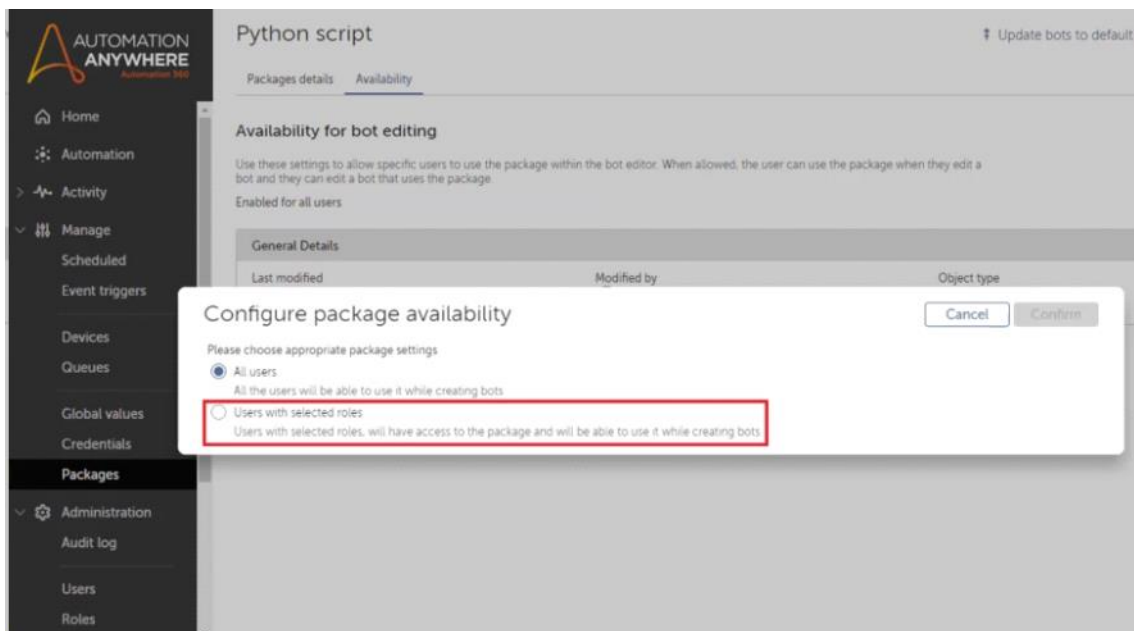
Role-Based Access Control for packages is performed by the Control Room Administrator or any user that has the following permissions:

- View Users and Roles basic information
- Manage packages

Let's consider a scenario where there are two groups of users - one that includes pro developers who require access to a package and another that includes Citizen developers that don't need access to that package.

The Control Room Administrator must perform the following steps to ensure appropriate RBAC for the package:

1. Create a custom role
2. To this role, add the pro developer users that need access to the package
3. In the Availability settings for the package, select the role that should have access to the package



In summary, you learned that:

- An Automation 360 Control Room Administrator can restrict the availability of selected action packages to specific user roles.
- Users that do not have access to selected action packages will not be able to use them while creating bots.
- However, such users can clone a bot created by other users who have access to the action package and run such bots.
- They however cannot make changes to such bots.

Useful Links

[Templates \(automationanywhere.com\)](https://automationanywhere.com)