**Environment control**

What is environment control?

Controlling the environment where your robots run is a major part of software robot development.

Ensuring that each run happens in a clean and repeatable environment, preferably without any local manual installations, is key to scaling and getting the most benefit from your software robot projects.

Robocorp environment control can be broken into two:

* Environment control using the package management system **[conda](https://docs.conda.io/en/latest/" \t "_blank)**
* Environment control that relies on local machine setup

Environment control using conda

Conda provides tools to create isolated environments that can use different versions of, for example, Python, Robot Framework, and a whole host of libraries. Conda also provides tools for "installing" these kinds of things without any operating system installations. This also protects the actual machine running from polluted environment variables etc.

Example: Having one robot that needs Python v2.7 and another that requires Python 3.8 can be a hassle to manage as the Windows PATH variable will provide the one that was installed last.

The way conda usage has been built into Robocorp tooling can be seen in the "robot template". The environment is defined in the conda.yaml file.

If the robot contains conda.yaml file, the environment is created according to the config given.

You can use the conda.yaml file to add conda and/or pip packages to your running environment. Both VS Code via RCC and Robocorp Workforce Agent use this config when running without any extra steps.

Each robot run via RCC or the Robocorp Workforce Agent gets a unique folder for execution, and the conda environment data is placed inside that folder. By default, Robocorp Workforce Agent holds the last five robot run folders. This means your runtime environment will not get cluttered with either run specific data or the environments used.

We heavily recommend this method, and the related supports for using conda are built-in to VS Code via RCC and Robocorp Workforce Agent.

Using this method, you can control [which packages and which versions of those packages your robot will use editing one single configuration file](https://robocorp.com/docs/setup/installing-python-package-dependencies).

Environment control that relies on local machine setup

Not all robots require Python, Robot Framework, etc. and robots sometimes need to heavily rely on specific machines installations and environments. In these cases, setting up the conda environments makes no sense.

The conda setup is optional in Robocorp products and can be skipped by removing the environment config file in the robot folder.

If the robot does NOT contain conda.yaml file, conda setup is skipped, and the run will see just the environment of the machine it is running on.

In these cases, you have to set up the targetted runtime environments for the robot.

Example: If your robot requires a specific zip tool to be available via Windows PATH you should install it and make sure the variable is set.

Each robot run is still getting a unique folder for the execution so that any temporary files created by the last run do not affect the next one.

See [Conda main site](https://docs.conda.io/projects/conda/en/latest/) for more information.

**robot.yaml**

tasks:

Run all tasks:

shell: python -m robot --report NONE --outputdir output --logtitle "Task log" tasks.robot

condaConfigFile: conda.yaml

artifactsDir: output

PATH:

- .

PYTHONPATH:

- .

ignoreFiles:

- .gitignore

**~~설치 순서~~**

1. ~~Install anaconda~~
2. ~~Install git client~~
3. ~~Install VS Code~~
4. ~~Setup VS Code~~

~~– install robocorp extensions~~

~~– Python Interpreter~~

1. ~~Create Project folder~~
2. ~~생성한 project folder에서 git clone 사용하여 github 내 project 다운로드~~

~~VS Code 내 clone 기능을 사용하면 추가적인 내용이 생성. 따라서 github에서 URL 복사하여 CMD 창에서 git clone 기능을 수행하는 것이 바람직~~

(Overview.docx에 추가 완료)

# Adding packages to your robot

One of the Robocorp automation stack advantages is that you can benefit from the extensive ecosystem of Robot Framework libraries and freely available Python packages.

When you create a new project using our development tools, you will get a ready-to-go base environment. The environment will include the [rpaframework](https://robocorp.com/docs/automation-libraries/rpa-framework-overview) package, giving you access to the RPA Framework set of open-source libraries supported and developed by Robocorp. You can add more features to your robot by adding more libraries and packages, or even remove the package if your code does not need it. Let's see how!

## A small introduction: conda and pip

The Robocorp automation stack uses the [conda](http://conda.io/" \t "_blank) open-source dependency management system to simplify the management of Python environments. Check the [environment control](https://robocorp.com/docs/setup/environment-control) page for more details.

As often happens in the open-source world, this is not the only solution available: another popular package management system for Python modules is [pip](https://pypi.org/project/pip/). Conda is a newer initiative, so not all packages are available for both systems. For this reason, we decided to support both systems.

### Should I use conda packages or pip packages?

Conda has a more efficient way to install packages and manage dependencies, which results in better and faster performance when setting up environments. So, **if the library or module you are interested in is available as a conda package, we recommend going with it**. As a general rule, only use the pip version of a package if it is not available in conda.

## The conda.yaml file

The conda.yaml file (see [default project structure](https://robocorp.com/docs/setup/robot-structure)) defines the environment your robot will run in. The default conda.yaml file you get when creating a new project looks something like this (comments added for clarity):

# Conda channels. We recommend using packages from the conda-forge channel.

channels:

- conda-forge

dependencies:

# Defining conda packages:

- python=3.7.5

# Adding pip itself as a conda package:

- pip=20.1

- pip:

# Defining pip packages:

- rpaframework==14.1.1

When given this configuration file, conda will:

* use the conda-forge conda [channel](https://docs.conda.io/projects/conda/en/latest/user-guide/concepts/channels.html)
* install a specific version of Python (which is available as a conda package)
* install a specific version of pip
* install the rpaframework pip package, at version 14.1.1.

Note: we are at work to be able to distribute rpaframework as a conda package as well.

## Where do I find packages?

Both systems provide web interfaces to search for supported packages:

* conda: <https://anaconda.org/search> (we recommend to choose packages from the conda-forge channel)
* pip: <https://pypi.org/>

## How do I add new packages?

Once you found the package you want to use, add it to the correct section of the conda.yaml file. This can be done manually or automated using [RCC](https://robocorp.com/docs/rcc/overview), our command-line tool.

### **Adding conda packages**

For example, if your robot requires the [numpy](https://pypi.org/project/numpy/" \t "_blank) **package**, which is available as a [conda package](https://anaconda.org/anaconda/numpy" \t "_blank), you can add it under the dependencies YAML attribute. In this case, we are specifying the latest version at the time of writing.

channels:

- conda-forge

dependencies:

- python=3.7.5

- pip=20.1

# Numpy added here:

- numpy=1.20.3

- pip:

- rpaframework==14.1.1

You can achieve the same result using RCC and its robot libs command:

rcc robot libs --add numpy=1.20.3 --conda conda.yaml

* Use the --add parameter to specify the name and version of the package.
* Use the --conda parameter to specify the path to the conda.yaml file.

### **Adding pip packages**

For example, if your robot requires the [robotframework-datadriver](https://github.com/Snooz82/robotframework-datadriver" \t "_blank) **Robot Framework library**, which is only available as a [pip package](https://pypi.org/project/robotframework-datadriver/), you will have to add it under the pip attribute:

channels:

- conda-forge

dependencies:

- python=3.7.5

- pip=20.1

- pip:

- rpaframework==14.1.1

# robotframework-datadriver added here:

- robotframework-datadriver==1.4.1

You can achieve the same result using RCC and its robot libs command:

rcc robot libs --pip --add robotframework-datadriver==1.4.1 --conda conda.yaml

* Use the --add parameter to specify the name and version of the package.
* Use the --conda parameter to specify the path to the conda.yaml file.
* Use the --pip parameter to indicate that you want to install a pip package.

# Robot YAML configuration format

## Robot YAML format

For a robot.yaml to be valid, it has only two mandatory parts:

* At least one task defined, and that must have either command, shell or robotTaskName defined.
* Artifacts output path must be defined.

**tasks:**

# You can define 1..n tasks to a robot.

# Naming: Think of actions or verbs this robot can perform.

# Task names given here are visible in Control Room.

# The task supports three ways of defining the action performed:

# `command`, `shell` or `robotTaskName`.

# Below are examples for each.

**User specified task name:**

# 'command': Separates the arguments to a list

# that takes care of arguments with spaces.

**command:**

- python

- -m

- robot

- --report

- NONE

- --outputdir

- output

- --logtitle

- Task log

- tasks.robot

**User specified task name 2:**

# 'shell': You have to quote items in the command with spaces using "

**shell:** python -m robot --report NONE --outputdir output --logtitle "Task log" tasks.robot

**User specified task name 3:**

# 'robotTaskName': Assumes a task with the same name exists in a .robot file.

**robotTaskName**: Calculate and log the result

**condaConfigFile:**

conda.yaml

# A relative path to your environment config file.

# Defining the conda.yaml file is optional.

# E.g., if the running environment is preset and you don't need any setup.

**artifactsDir:**

output

# A relative path to a folder where the artifacts are stored.

# The contents of this folder will be sent to Control Room.

**PATH:**

# The paths listed here are added to the PATH environment variable

# for the duration of the execution.

- .

**PYTHONPATH:**

# The paths listed here are added to the PYTHONPATH environment variable

# for the duration of the execution.

- .

**ignoreFiles:**

# A relative path to the .gitignore file that controls what is placed in the

# robot zip file. This can be used to control what items are not packaged

# when pushing the robot to Control Room. Defining this is optional.

- .gitignore

**Note that with the robot.yaml you can define:**

* A large robot with multiple Robot Framework tasks or tests and a complex environment.
* A simple Python robot with some environment requirements.
* Just a simple executor that runs a known script in a preset environment.

...and everything in between, examples below.

## Examples

### **A simple single task robot example**

An example where you only have a single Robot Framework file to execute with minimal folder structure and no optional fields:

tasks:

Read the Emails:

shell: python -m robot --report NONE --outputdir output --logtitle "Task log" tasks.robot

condaConfigFile: conda.yaml

artifactsDir: output

PATH:

- .

PYTHONPATH:

- .

### **Full example with multiple tasks**

tasks:

Read Input Forms:

robotTaskName: Read Inputs

Create PDFs:

robotTaskName: Generate PDF

Email Customers:

robotTaskName: Send Emails

condaConfigFile: conda.yaml

artifactsDir: output

ignoreFiles:

- .gitignore

PATH:

- .

- bin

PYTHONPATH:

- .

- variables

- libraries

- resources

### **Minimal example**

An example, where you only have scripts to execute, you don't need Robot Framework nor any environment setup. In this case, your robot can be just a robot.yaml file and nothing else.

tasks:

Trigger start script:

shell: C:\\my-known-location\\my-trigger-script.bat

artifactsDir: output

PATH:

- .