spade-artifact Documentation

Release 0.1.1

Javi Palanca

Jul 05, 2021

CONTENTS:

CHAPTER

ONE

SPADE-ARTIFACT

Plugin for SPADE 3 to develop artifacts.

- Free software: MIT license
- Documentation: https://spade-artifact.readthedocs.io.

1.1 Features

- Focus on observable properties (based on PubSub).
- Presence notifiaction support.
- Artifact class.
- ArtifactMixin for SPADE agents.

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install spade-artifact, run this command in your terminal:

```
$ pip install spade_artifact
```

This is the preferred method to install spade-artifact, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for spade-artifact can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/javipalanca/spade_artifact
```

Or download the tarball:

```
$ curl  -OL https://github.com/javipalanca/spade_artifact/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

Note: This is a plugin for the SPADE agent platform. Please visit the SPADE's documentation to know more about this platform.

To use spade-artifact in a project you can follow the next example, which implements an artifact that periodically emits random numbers only if any contact is online:

```
import asyncio
import random
import getpass

from loguru import logger
from spade.agent import Agent

from spade_artifact import Artifact, ArtifactMixin


class RandomGeneratorArtifact(Artifact):
    def on_available(self, jid, stanza):
        logger.success(
            "[{}] Agent {} is available.".format(self.name, jid.split("@")[0])
        )

    def on_subscribed(self, jid):
        logger.success(
            "[{}] Agent {} has accepted the subscription.".format(
                self.name, jid.split("@")[0]
            )
        )
        logger.success(
            "[{}] Contacts List: {}".format(self.name, self.presence.get_contacts())
        )

    def on_subscribe(self, jid):
        logger.success(
            "[{}] Agent {} asked for subscription. Let's aprove it.".format(
                self.name, jid.split("@")[0]
            )
        )
        self.presence.approve(jid)
```

(continues on next page)

(continued from previous page)

```
        self.presence.subscribe(jid)

    async def setup(self):
        # Approve all contact requests
        self.presence.set_available()
        self.presence.on_subscribe = self.on_subscribe
        self.presence.on_subscribed = self.on_subscribed
        self.presence.on_available = self.on_available

    async def run(self):

        while True:
            # Publish only if my friends are online
            if len(self.presence.get_contacts()) >= 1:
                random_num = random.randint(0, 100)
                await self.publish(f"{random_num}")
                logger.info(f"Publishing {random_num}")
            await asyncio.sleep(1)


class ConsumerAgent(ArtifactMixin, Agent):
    def __init__(self, *args, artifact_jid: str = None, **kwargs):
        super().__init__(*args, **kwargs)
        self.artifact_jid = artifact_jid

    def artifact_callback(self, artifact, payload):
        logger.info(f"Received: [{artifact}] -> {payload}")

    async def setup(self):
        await asyncio.sleep(2)
        self.presence.approve_all = True
        self.presence.subscribe(self.artifact_jid)
        self.presence.set_available()
        await self.artifacts.focus(self.artifact_jid, self.artifact_callback)
        logger.info("Agent ready")


if __name__ == "__main__":
    XMPP_SERVER = input("XMPP Server>")
    artifact_jid = f"{input('Artifact name> ')}@{XMPP_SERVER}"
    artifact_passwd = getpass.getpass()

    agent_jid = f"{input('Agent name> ')}@{XMPP_SERVER}"
    agent_passwd = getpass.getpass()

    agent = ConsumerAgent(
        jid=agent_jid, password=agent_passwd, artifact_jid=artifact_jid
    )
    agent.start()

    artifact = RandomGeneratorArtifact(artifact_jid, artifact_passwd)
```

(continues on next page)

(continued from previous page)

```
    future = artifact.start()
    future.result()

    artifact.join()
```

The example below shows the main features required to build an artifact and to interact with artifacts as an agent. As shown, an artifact MUST implement its `run` coroutine where its main functionality is presented (some initial configuration can still be done from the `setup` coroutine).

An artifact can publish observations by means of the `publish` coroutine as shown in the example.

Also, an artifact can handle presence messages using the same API as a SPADE agent.

On the other hand, an agent can interact with artifacts just by inheriting from the ArtifactMixin class. It provides the necessary stuff to be able to focus on an artifact and receive its observations. As in the example below, an agent can use the `self.artifacts.focus` coroutine to focus on an artifact. The parameters are the jid of the artifact and the callback method that will receive the observations. This callback method will receive as arguments the jid of the artifact publishing the observation and the payload of the observation.

Warning: Remember that, when inheriting from Mixins, they MUST be always before the base class (`Agent`). E.g. `class MyAgent(PubSubMixin, ArtifactMixin, Agent):`

If an agent wants to stop focusing on an artifact it can use the `self.artifacts.ignore` coroutine with the jid of the artifact.

CHAPTER

FOUR

API DOCUMENTATION

4.1 spade_artifact package

4.1.1 Submodules

4.1.2 spade_artifact.agent module

class spade_artifact.agent.**ArtifactComponent**(*agent*)
Bases: object

async focus(*artifact_jid*, *callback*)

async ignore(*artifact_jid*)

on_item_published(*jid*, *node*, *item*, *message=None*)

class spade_artifact.agent.**ArtifactMixin**(**args*, *pubsub_server=None*, ***kwargs*)
Bases: spade_pubsub.pubsub.PubSubMixin

4.1.3 spade_artifact.artifact module

class spade_artifact.artifact.**Artifact**(*jid*, *password*, *pubsub_server=None*, *verify_security=False*)
Bases: spade_pubsub.pubsub.PubSubMixin

get(*name*)
Recovers a knowledge item from the artifact's knowledge base.

Args: name(str): name of the item

Returns: object: the object retrieved or None

is_alive()
Checks if the artifact is alive.

Returns: bool: wheter the artifact is alive or not

join(*timeout=None*)

kill()

mailbox_size() → int
Checks if there is a message in the mailbox

Returns: int: the number of messages in the mailbox

property name
Returns the name of the artifact (the string before the '@')

async publish(*payload: str*) → None

async receive(*timeout: Optional[float] = None*) → Optional[spade.message.Message]
Receives a message for this artifact. If timeout is not None it returns the message or "None" after timeout is done.

Args: timeout (float): number of seconds until return

Returns: spade.message.Message: a Message or None

async run()
Main body of the artifact. This coroutine SHOULD be overloaded.

async send(*msg: spade.message.Message*)
Sends a message.

Args: msg (spade.message.Message): the message to be sent.

set(*name*, *value*)
Stores a knowledge item in the artifact knowledge base.

Args: name (str): name of the item value (object): value of the item

set_container(*container*)
Sets the container to which the artifact is attached

Args: container (spade.container.Container): the container to be attached to

set_loop(*loop*)

async setup()
Setup artifact before startup. This coroutine may be overloaded.

start(*auto_register=True*)
Tells the container to start this agent. It returns a coroutine or a future depending on whether it is called from a coroutine or a synchronous method.

Args: auto_register (bool): register the agent in the server (Default value = True)

stop()
Stop the artifact

4.1.4 Module contents

Top-level package for spade-artifact.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/javipalanca/spade_artifact/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

spade-artifact could always use more documentation, whether as part of the official spade-artifact docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/javipalanca/spade_artifact/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *spade_artifact* for local development.

1. Fork the *spade_artifact* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/spade_artifact.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv spade_artifact
$ cd spade_artifact/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 spade_artifact tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/javipalanca/spade_artifact/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_spade_artifact
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Javi Palanca <jpalanca @ gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

HISTORY

7.1 0.1.1 (2021-07-05)

- Fixed documentation style.

7.2 0.1.0 (2021-07-05)

- First release on PyPI.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX