This Document Provides the overview of the surround framework explaining all the features and sub-features of surround. The document is not complete; hence the full functionality of the framework cannot be explained through this document.



Surround AI

# User Manual

Version 1.1

18/05/2019

Document Number: 1.1

Table of Contents

[1. Introduction 1](#_Toc9090312)

[2. Overview 2](#_Toc9090313)

[2.1 Why Surround? 2](#_Toc9090314)

[3. Getting Started 3](#_Toc9090315)

[3.1 Requirements: 3](#_Toc9090316)

[3.2 Mac Os: 3](#_Toc9090317)

[3.3 Windows: 4](#_Toc9090318)

[4. Components of Surround 6](#_Toc9090319)

[4.1 Pythonic Interface 7](#_Toc9090320)

[4.2 Classes of Surround 8](#_Toc9090321)

[5. Troubleshooting & Support 12](#_Toc9090322)

[5.1 Error Messages 12](#_Toc9090323)

[5.2 Special Considerations 12](#_Toc9090324)

[5.3 Support 12](#_Toc9090325)

[Appendix A: Record of Changes 13](#_Toc9090326)

[Appendix B: Acronyms 14](#_Toc9090327)

[Appendix C: Glossary 15](#_Toc9090328)

[Appendix D: Referenced Documents 16](#_Toc9090329)

[Appendix E: Approvals 17](#_Toc9090330)

List of Figures

**No table of figures entries found.**

List of Tables

[Table 1 - Support Points of Contact 12](#_Toc9090391)

[Table 2 - Record of Changes 13](#_Toc9090392)

[Table 3 - Acronyms 14](#_Toc9090393)

[Table 4 - Glossary 15](#_Toc9090394)

[Table 5 - Referenced Documents 16](#_Toc9090395)

[Table 6 - Approvals 17](#_Toc9090396)

## Introduction

This User Manual (UM) provides the information necessary for Data Scientists and Software Engineers to effectively use the Surround AI.

The framework of Surround has been primarily designed and developed to assist the Data-scientist who utilizes the python platform to perform their extraction and interpretation of Big-data. The Data-scientists usually spends a lot of time in the process of collection, cleaning and filtering data into datasets. This requires a lot of effort and time consumption. This is where this framework becomes extremely useful, not just because of its user-friendliness, but also it saves hours, the Data-scientist spends on writing the codes.

Surround is a framework used for machine learning pipelines which are used to help automate the work flow in terms of cleaning, visualizing and interpreting data within the analysis in the python script. The framework is flexible and developed in a way that it could help the data scientists to render the problem within their scope rather than understanding any cryptic code.

## Overview

Surround is a lightweight framework for serving machine learning pipelines in Python.

1. It is intended to be adaptable and simple to utilize.

2. It helps Data scientist by letting them concentrate on the current issue instead on the development of code.

3. It is still being under constant and continuous development and expansion.

**2.1 Audience:**

The framework of Surround has been primarily designed and developed to assist the Data-scientist who utilizes the python platform to perform their extraction and interpretation of Big-data. The Data-scientists usually spends a lot of time in the process of collection, cleaning and filtering data into datasets. This requires a lot of effort and time consumption. This is where this framework becomes extremely useful, not just because of its user-friendliness, but also it saves hours, the Data-scientist spends on writing the codes.

**2.2 Features:**

Surround framework was developed for Artificial intelligence to address the following issues.

1. Continuous restructuring of existing code before implementation. It is intended to improve non-functional attributes of the code.

2. No standardization for the configuration handling, pipeline architecture and building scripts.

3. Failure to provide and end-to-end serving solution between the models.

4. Existing serving approaches don’t take into consideration the development of an AI pipeline without re-designing the arrangement.

5. Code was commonly being commented out to run other branches as experimentation was not a first-class citizen in the code being written.

### Why Surround?

* A flexible way to serve a pipeline in Python without writing C/C++ code.
* Multiple models (custom or pre-trained) from different frameworks that need to be combined into a single Surround solution.
* To use an existing intelligent APIs (AWS Recognition, Google Cloud AI, Cognitive Services) as part of your Surround implementation.
* You have pre or post processing steps that aren't part of your models but need to be deployed as part of your Surround implementation.
* You need to package up your dependencies for running Surround as an offline solution on another machine.

## Getting Started

### Requirements:

* Intel Core i5 processor 4300M at 2.60 GHz or 2.59 GHz, 8 GB of DRAM
* Intel Xeon processor E5-2698 v3 at 2.30 GHz, 64 GB of DRAM
* Intel Xeon Phi processor 7210 at 1.30 GHz, 32 GB of DRAM, 16 GB of MCDRAM.
* Disk space: 2 to 3 GB
* Operating systems: Windows 10, MacOS, and Linux

**Minimum System Requirements**

* Processors: Intel Atom processor or Intel Core i3 processor
* Disk space: 1 GB
* Operating systems: Windows7 or later, MacOS, and Linux
* Python versions: 3 or above
* Included development tools: conda\*, conda-env, Jupyter Notebook\* (I Python)
* Compatible tools: Microsoft Visual Studio\*, PyCharm\*
* Included Python packages: NumPy, SciPy, scikit-learn\*, pandas, Matplotlib, Numba\*, Intel Threading Building Blocks, pyDAAL, Jupyter, mpi4py, PIP\*, and others.

### Mac Os:

First, check whether the python is installed in our system using terminal with the command: python --version. If we cannot see the installed version or the version of python below 3.0 then uninstall the version present in the system and follow the steps below.  It includes two stages i.e. installation of homebrew and installation of python

**Step 1**: **installation of home brew**

* Navigate to the link <http://brew.sh/>. Select and copy the homebrew bootstrap code.
* Open the terminal app window and paste the homebrew bootstrap code.
* Now the homebrew installation process begins.
* Assign user account password to the homebrew to complete the installation process.

**Step 2**: **installation of python 3**

* Open the terminal and run the following command: brew install python3
* The above command will download and install the latest version of python.
* Finally, python 3 should be successfully installed in the system
* Testing process of installation:
* Open the terminal app.
* Type pip3. This command helps to test whether the python is correctly installed in the system.
* If an error occurs when running pip3, go through the python install steps and make sure you have a working python installation.

### Windows:

First step is to check whether the python is installed earlier or not. Open the command and use the command python –version.  If there is an installed python version below 3.0 uninstall python and follow the steps to install the higher version of python.  You can download the python latest version using the link:

<https://www.python.org/downloads/>.

* Double click the downloaded file to run the file python-3.7.3.
* Python 3.7.3 (32-bit) setup pop-up window will appear.
* Install launcher for all users (recommended) and add python 3.7 to PATH checkboxes are checked in that pop-up window.
* Click the Install Now option in the pop-up window to start the installation process.
* A user account control pop-up window will appear, posing the question Do you want the allow the following program to make changes to this computer?
* Click yes option displayed in that pop-up window.
* A new Python 3.7.0 (32-bit) Setup pop-up window will appear with a Setup Progress message and a progress bar.
* After progress bar is moved towards the completion, new pop-up window will appear showing Python 3.7.0 (32-bit) Setup was successfully installed.

**3.3.1 Verifying the installation process:**

* Navigate to the directory where ever the Python is installed.
* Double click the file python.exe.
* A pop-up window appears; inside the window it shows the version of Python.
* It indicates that Python is installed successfully in the system.

**3.3.2 Installation of surround:**

* Open the terminal and use the command: pip3 install surround.
* The above command directly installs the surround.
* To run the tests, we can use command: python3 setup.py test.
* A sample example Hello world will be executed showing that surround is executed successfully.
* Next step after installation of surround is project creation.
* To create a project, we can use the command: surround init <path-to-dir>
* To assign the project name and description of the created project we can use: surround init <path-to-dir> --project-name –description.

**IDE setup:**

* The IDE used in our project is PyCharm.
* Download and install PyCharm.
* PyCharm can be downloaded using the link: <https://pycharm.en.softonic.com/>
* Once the download is completed double click the downloaded file then the installation process begins.
* After successful installation of PyCharm then select the interpreter as Anaconda.

## Components of Surround

The below mentioned components are in this library that can be utilized to build surround solution.

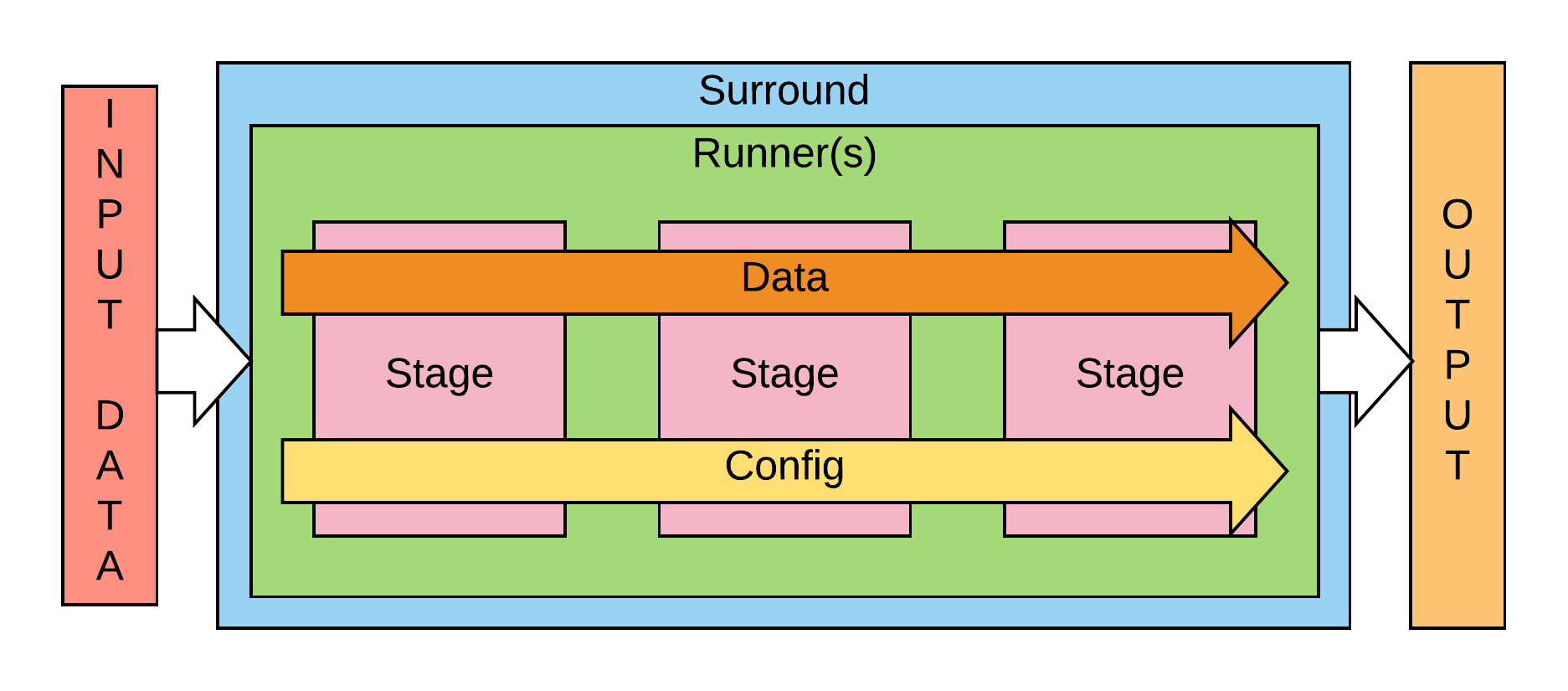
1. Surround

2. Surround Data

3. Stage

4. Runner

* 1. **Surround**: It is a group of numerous stages or just an initial stage to change raw information into meaningful data. You can set the order of stages directly or by means of a config file. The config file enables you to characterize more than 1 pipeline execution and after that you can switch between them effectively.
  2. **Surround Data**: A sharable item between stages that holds vital data for each stage. A phase will read some data from Surround Data, process it, at that point set back new data that will be utilized by different stage(s). When you broaden this class, you can include as many numbers of variables as you require to enable you to change input data into output data. In any case, there are 4 center factors that are being utilized.
* stage\_metadata is information that can be used to identify a stage.
* execution\_time is recorded time to complete a process.
* errors are information to identify failure of a stage.
* warnings are information when transformation is not 100% right.
  1. **Stage**: A usage of information change. Here is the place Surround Data is altered to accomplish the outcome that you need. Each stage is just meant to execute out a lot of related actions. First stage can be where you get ready information to be prepared and last stage can be the place you populate information to be sent back to the client.
     + **operate** is a function that you need to override when you extend stage class. It should contain data transformation implementation
  2. **Runner**: (optional) An interface to connect Surround to/from data.



Fig(a) Surround framework components

### Pythonic Interface

Surround has a pythonic interface and allows the developer to utilize this framework just as any of the already existed modules. This makes it a powerful framework of data-exploration purposes.

The common example of Hello World program implementation on surround framework is given below.

*from surround import Stage, SurroundData, Surround*

*import logging*

*classHelloStage(Stage):*

*defoperate(self, data, config):*

*data.text =&quot;hello&quot;*

*classBasicData(SurroundData):*

*text =None*

*if\\_\\_name\\_\\_==&quot;\\_\\_main\\_\\_&quot;:*

*logging.basicConfig(level=logging.INFO)*

*surround = Surround([HelloStage()])*

*output = surround.process(BasicData())*

*print(output.text)*

Let us now understand how the code works;

* + The example prints the text &quot;hello&quot; utilizing surround.
  + The process consists of defining the operation in the perate() method of HelloStage.
  + The object being processed is an instance of BasicData, which inherits from
  + he surround object is initialized with only one stage as surround = Surround([HelloStage()], and the line output = surround.process(BasicData()) calls the operate() method of HelloStage.
  + New instance BasicData is used as a parameter and finally its contents are printed to the screen.

### Classes of Surround

This part is dedicated to explaining the different Classes of Surround.

Classes run a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods defined by its class for modifying its state.

**Let us see some of the classes**

1. class Config (Mapping):
2. class LinterStage (Stage):
3. class Stage (ABC):
4. class Frozen ():
5. class SurroundData (Frozen):
6. class Surround (ABC):
7. class AllowedTypes (Enum):
8. class Wrapper ():

**4.2.1 class Config (Mapping):**

This class helps in Identifying the different paths for output, data, models, Importing the location.

**The main features of mapping include:**

1. creating continuous return series for Futures instruments
2. creating time series of percentage allocations to tradeable contracts
3. creating instrument trade lists.
4. Use models’ paths to maintain consistency

class Config (Mapping):

def \_\_init\_\_ (self, project\_root=None):

    self.\_storage = self.\_\_load\_defaults()

    # Set framework paths

    if project\_root:

        self.\_storage["project\_root"] = project\_root

        self.\_storage["output\_path"] = os.path.join(project\_root, "output")

        self.\_storage["data\_path"] = os.path.join(project\_root, "data")

        self.\_storage["models\_path"] = os.path.join(project\_root, "models")

**4.2.2 class LinterStage (Stage):**

This class helps in verifying the quality of the code and analyse source code for potential errors, lint is a tool which is used to mark the source code with some suspicious and non-structural.

**These classes Add support for the Running Project specific tasks**

1. For the Lint process to execute and check for errors and add warning if bugs are found,
2. class LinterStage (Stage):
3. To check the Data in the linter stage
4. class CheckData(LinterStage):
5. To check the Surround project files
6. class CheckFiles(LinterStage):
7. To check the Directories and validating Surround's directory structure
8. class CheckDirectories(LinterStage):
9. To check the Project Details, i.e project\_structure, project\_root, project\_name.
10. class ProjectData(SurroundData):
11. To check the Surround files, Data, Directories
12. class Linter():
13. linter\_checks = Surround([CheckDirectories(), CheckFiles(), CheckData()])

**4.2.3class Stage (ABC):**

It replaces and Stores intermediate data from each stage in the pipeline to the Dump output of each stage.

class Stage(ABC):

def dump\_output(self, surround\_data, config):

**For Example:**

 from abc import ABC, abstractmethod

 class C(ABC):

 @abstractmethod

 def my\_abstract\_(cls, ...):

    ...

**4.2.4 class Frozen ():**

This class can toggle the ability of adding new attributes. Frozen classes are, then, classes which can't be changed after they've been created.

To have a fully frozen class, all attributes of the class should hold immutable values too.

 """

\_\_isfrozen = False

def \_\_setattr\_\_(self, key, value):

    if self. \_\_isfrozen and not hasattr (self, key):

        raise TypeError("%r is a frozen object" % self)

    object. \_\_setattr\_\_(self, key, value)

**4.2.5 class SurroundData (Frozen):**

Stores the data to be passed between each stage in Surround. Considering the frozen data as different stages inside the Surround are responsible for setting the attributes.

     """

    stage\_metadata = []

    execution\_time = None

    errors = []

    warnings = []

**4.2.6 class Surround (ABC):**

The collections module has some concrete classes that derive from ABCs; these can be further derived. In addition, the collections. ABC sub-module has some ABCs that can be used to test whether a class or instance gives an interface.

   def \_\_init\_\_ (self, surround\_stages=None, module=None):

     self.surround\_stages = surround\_stages

**4.2.7 class AllowedTypes (Enum):**

An Enum is a set of symbolic names or members bound to unique, constant values. Within an enumeration, the members can be compared by identity, and the Enum itself can be repeated over and over.

   JSON = ["application/json"]

   FILE = ["file"]

**4.2.8 class Wrapper ():**

This is a wrapper class which, it wraps an object which it then proxies unhandled getter calls to Wrapper functions can be used as an interface to adapt to the existing codes, to save you from changing your current codes back and forth.

    def \_\_init\_\_(self, surround, type\_of\_uploaded\_object=None):

    self.surround = surround

    self.actual\_type\_of\_uploaded\_object = None

    if type\_of\_uploaded\_object:

        self.type\_of\_uploaded\_object = type\_of\_uploaded\_object

    else:

        self.type\_of\_uploaded\_object = AllowedTypes.JSON

    self.surround.init\_stages()

## Troubleshooting & Support

Instructions: Describe all recovery and error correction procedures, including error conditions that may be generated and corrective actions that may need to be taken. Organize the information in sub-sections as appropriate. The following are common sub-sections that may be included as appropriate.

### Error Messages

Instructions: Identify the error messages that a user may receive and the likely cause(s) and/or possible corrective actions for the error. If the list is extensive, this information may be best provided in an appendix to the document that is referenced here.

### Special Considerations

Instructions: If applicable, describe any special circumstances, actions, caveats, exceptions, etc., that should be considered for troubleshooting.

### Support

Table 1 - Support Points of Contact

| Contact | Organization | Email | Role |
| --- | --- | --- | --- |
| Akshat Bajaj | Applied Artificial Intelligence Institute | [Akshat.bajaj@deakin.edu.au](mailto:Akshat.bajaj@deakin.edu.au) | Software Engineer |
| Thanh Nguyen | Deakin University | [Duc.nguyen@deakin.edu.au](mailto:Duc.nguyen@deakin.edu.au) | Supervisor |

## Frequently Asked Questions

1. **Why Surround in Python?**

 Python does not need to be compiled into machine language instruction before execution and can be used by the developer directly to run the program, Python offers the least code among others and is in fact 1/5 the number compared to other OOP languages. No wonder it is one of the most popular in the market today.

Python has Prebuilt Libraries like Numpy for scientific computation, Scipy for advanced computing and Pybrain for machine learning (Python Machine Learning) making it one of the best languages for AI.

Python is platform Independent and is hence one of the most flexible and popular choices for use across different platforms and technologies with the least tweaks in basic coding.

Python is the most flexible of all others with options to choose between OOPs approach and scripting. You can also use IDE itself to check for most codes and is a boon for developers struggling with different algorithms.

1. **What are machine learning pipelines?**

Machine Learning pipelines are used to carry the code from developers to the data scientists and make the code ready before initialising the code to the data scientists with the help of machine learning.

1. **Why this Surround AI Model?**

Existing model serving solutions focus on serving the model rather than serving an end-to-end solution. Our machine learning projects require multiple models and glue code to tie these models together.

1. **What are stages in Surround AI?**

There are three stages in surround AI Framework. Which are used to transform the data from one to another after transforming into more meaningful information.

1. **What is**[**BSD-3**](https://opensource.org/licenses/BSD-3-Clause)**license?**

BSD licenses are the family of permissive free software licenses, imposing minimal restrictions on the use and distribution of covered software. BSD-3 license allows almost unlimited freedom with the software so long as you include the BSD copyright and license notice in it.

Appendix A: Record of Changes

Table 2 - Record of Changes

| Version Number | Date | Author/Owner | Description of Change |
| --- | --- | --- | --- |
| 1.1 | 18/05/2019 | Unique Poudel | User Manual Created |
|  |  |  |  |
|  |  |  |  |

Appendix B: Acronyms

Table 3 - Acronyms

| Acronym | Literal Translation |
| --- | --- |
| AI | Artificial Intelligence |
|  |  |
|  |  |

Appendix C: Glossary

Table 4 - Glossary

| Term | Acronym | Definition |
| --- | --- | --- |
| <Term> | <Acronym> | <Definition> |
| <Term> | <Acronym> | <Definition> |
| <Term> | <Acronym> | <Definition> |

Appendix D: Referenced Documents

Table 5 - Referenced Documents

| Document Name | Document Location and/or URL | Issuance Date |
| --- | --- | --- |
| Readme.md | <https://github.com/a2i2/surround/blob/master/README.md#when-to-use-surround> | 18/05/2019 |
| Template |  |  |
|  |  |  |

Appendix E: Approvals

The undersigned acknowledge that they have reviewed the User Manual and agree with the information presented within this document. Changes to this User Manual will be coordinated with, and approved by, the undersigned, or their designated representatives.

Table 6 - Approvals

| Document Approved By | Date Approved |
| --- | --- |
| Name: Thanh Nguyen, Supervisor – Surround AI Squad 2 | Date |
| Name: Akshat Bajaj, product owner- Surround AI Tribe | Date |
| Name: Mohamed Abdelrazek , Unit Chair- SIT 782 | Date |
| Name: Alessio Bonti, Unit chair – SIT 764 | Date |