ScienceLogic Libraries and Execution Environments

Skylar One version 12.5.20

Table of Contents

Introduction to ScienceLogic Libraries and Execution Environments ... **3**

What is a ScienceLogic Library? ... 4

What is an Execution Environment? ... 5

Managing ScienceLogic Libraries ... **7**

Viewing the List of ScienceLogic Libraries ... 8

Downloading a ScienceLogic Library ... 9

Importing a ScienceLogic Library ... 9

Creating a ScienceLogic Library ... 9

Important Notes on Creating ScienceLogic Libraries ... 10

Library File Type: py_package ... 11

Manually Building a ScienceLogic Library ... 12

Code Example ... 16

Converting a py_directory ScienceLogic Library ... 19

Deleting a ScienceLogic Library ... 22

Managing Execution Environments ... **24**

Viewing the List of Execution Environments ... 25

Creating an Execution Environment ... 25

Editing an Execution Environment ... 26

Copying an Execution Environment ... 27

Deleting an Execution Environment ... 27

Chapter 1

Introduction to ScienceLogic Libraries and Execution Environments

Overview

This manual describes how to create and manage ScienceLogic Libraries and Execution Environments in Skylar One.

This chapter covers the following topics:

What is a ScienceLogic Library? 4

What is an Execution Environment? 5

What is a ScienceLogic Library?

Some Dynamic Applications, credential tests, and Run Book Actions in Skylar One collect data or complete automated actions by executing one or more blocks of Python code, called ***snippets***. When this happens, Skylar One passes credentials and other configuration information to the snippet, and at the end of execution, the snippet either passes collected data back to Skylar One or runs an action.

A ***ScienceLogic Library*** is a package consisting of metadata and Python files that can be used by the Dynamic Applications, credential tests, and Run Book Actions that use snippets. ScienceLogic Libraries can be included in a PowerPack along with the Dynamic Applications, credential tests, and Run Book Actions that they support.

NOTE: ScienceLogic Libraries are built using Python 3.

CAUTION: Python 2 support was deprecated in Skylar One 12.5.1. If you still use Python 2 for custom code, you cannot upgrade to 12.5.1 or later releases until your custom code is Python 3-compatible.

These are a few reasons you might want to create and use a ScienceLogic library in your PowerPack. For example, if you have a Python package you created that you want to share between snippets, or you have a third-party package that needs to be available during snippet execution.

ScienceLogic Libraries use the py_package (.tar) format. This file type includes metadata and Python packages, and allows you to take advantage of unit tests, automatic versioning, and other common packaging practices. This format requires an understanding of Python packaging tools.

NOTE: Older versions of Skylar One bundled ScienceLogic Libraries in a py_directory format. That format has since been deprecated. Current ScienceLogic Libraries should be bundled only in a py_package format.

ScienceLogic Libraries can be installed on Skylar One in three ways:

- Some libraries are included when you install Skylar One or run system updates.
- Libraries that are included in a PowerPack are installed on Skylar One when you install the PowerPack.
- Libraries can be manually imported into Skylar One.

You can view all of the installed ScienceLogic Libraries on the **ScienceLogic Library Manager** page (System > Customize > ScienceLogic Libraries).

For details about how to manage libraries in Skylar One, see the *Managing ScienceLogic Libraries* chapter.

What is an Execution Environment?

To use a ScienceLogic Library, you must align it with an execution environment.

In the Skylar One user interface, an ***execution environment*** is a list of one or more ScienceLogic libraries. When a Dynamic Application snippet, Run Book Automation snippet, or credential test is executed, the execution environment defines a virtual Python environment that is deployed on-demand and includes all of the ScienceLogic libraries aligned with it for use during snippet execution.

A ScienceLogic Library can be associated with multiple execution environments, allowing you to share code between snippets, Dynamic Applications, and PowerPacks.

Execution environments must be specified when creating or editing the following:

- Snippets
- Credential tests
- Run Book Action policies that execute a snippet

When Skylar One runs a snippet embedded in a Dynamic Application, credential test, or a Run Book Action, it automatically deploys the runtime environment defined by the execution environment. The runtime environment includes each library aligned with the execution environment, plus system libraries that are provided by Skylar One.

If you do not specify the execution environment for a Dynamic Application, credential test, or Run Book Action, Skylar One aligns them with a system-default execution environment. This default execution environment is defined in each Skylar One System update.

When you build a PowerPack, it automatically includes the execution environments that are aligned with the PowerPack's Dynamic Applications, credential tests, and Run Book Actions that execute snippets.

NOTE: Additionally, built-in, non-snippet Dynamic Application types use a pre-deployed, read-only Python 3 execution environment, "sl1_default_ee". This execution environment appears on the **Environment Manager** page (System > Customize > ScienceLogic Libraries > Actions > Execution Environments), but you cannot edit or delete it. Whenever new built-in Dynamic Applications are created via the user interface, they are automatically assigned to this execution environment.

NOTE: The internal collection processes and Internal Collection Dynamic Applications (ICDAs) run on Python 3. The snippet code that is included in these Dynamic Applications must be Python 3.

CAUTION: Python 2 support was deprecated in Skylar One 12.5.1. If you still use Python 2 for custom code, you cannot upgrade to 12.5.1 or later releases until your custom code is Python 3-compatible.

For details about how to create and manage execution environments in Skylar One, see the *Managing Execution Environments* chapter.

For more information about aligning execution environments with Dynamic Applications, credential tests, or Run Book Actions, see the following manuals:

- *Dynamic Application Development*
- *Snippet Dynamic Application Development*
- *Internal Collection Dynamic Application Development*
- *Discovery and Credentials*
- *Run Book Automation*

Chapter 2

Managing ScienceLogic Libraries

Overview

This chapter describes how to manage ScienceLogic Libraries in Skylar One.

This chapter covers the following topics:

Viewing the List of ScienceLogic Libraries 8

Downloading a ScienceLogic Library 9

Importing a ScienceLogic Library 9

Creating a ScienceLogic Library 9

Manually Building a ScienceLogic Library 12

Converting a py_directory ScienceLogic Library 19

Deleting a ScienceLogic Library 22

Viewing the List of ScienceLogic Libraries

The **ScienceLogic Library Manager** page (System > Customize > ScienceLogic Libraries) displays a list of all available ScienceLogic Libraries. From this page, you can download libraries, import new libraries, delete existing libraries, or view and edit a list of Execution Environments that you can associate with a library.

For each library, the page displays the following information:

TIP: To sort the list of ScienceLogic Libraries, click on a column heading. The list will be sorted by the column value, in ascending order. To sort by descending order, click the column heading again. The ***Last Edited*** column sorts by descending order on the first click; to sort by ascending order, click the column heading again.

- ***Library Name***. The name of the ScienceLogic Library.
- ***Version***. The version number of the ScienceLogic Library.
- ***System***. Displays **True** if the ScienceLogic Library is included in the latest release of Skylar One and is aligned with the "System" environment. Displays **False** if the ScienceLogic Library is included in an imported PowerPack or manually installed by an administrator.
- ***Type***. The file type used to bundle and deliver the ScienceLogic Library. Supported types include the following:
    - *py_package (.tar format)*. A py_package is a tar file that contains Python packages and metadata files (name, version, description, etc.). The py_package format is typically used by more experienced developers to build complex solutions. ScienceLogic recommends using the py_package format because it is easier to test and maintain.
    - *py_directory (.tar format)*. A py_directory is a tar file that includes Python files and is typically used by novice developers to share short code snippets between Dynamic Applications or Skylar One stacks. This format does not let you manage items such as dependencies, version, or test, and is not recommended.

      NOTE: Older versions of Skylar One bundled ScienceLogic Libraries in a py_directory format. That format has since been deprecated. Current ScienceLogic Libraries should be bundled only in a py_package format.

- ***Dependencies***. A comma-separated list of other ScienceLogic Libraries on which the selected library is dependent, if applicable. These additional libraries are included in the selected library's *py_package.tar* file.
- ***Description***. A single-line description of the ScienceLogic Library's intended use.
- ***Python Version***. The Python compatibility version.
- ***Edited By***. The name of the ScienceLogic user who created or last edited the ScienceLogic Library.
- ***Last Edited***. The date and time the ScienceLogic Library was created or last edited.

Downloading a ScienceLogic Library

From the **ScienceLogic Library Manager** page, you can download ScienceLogic Libraries. Libraries that are included in Skylar One are bundled in a py_package format.

WARNING: ScienceLogic-authored libraries are copyrighted content and should not be modified without the express permission of ScienceLogic.

NOTE: Older versions of Skylar One bundled ScienceLogic Libraries in a py_directory format. That format has since been deprecated. Current ScienceLogic Libraries should be bundled only in a py_package format.

To download a ScienceLogic Library:

1. Go to the **ScienceLogic Library Manager** page (System > Customize > ScienceLogic Libraries).
2. Click the save icon ( ) of the ScienceLogic Library you want to download.
3. The library file is downloaded to your computer.

Importing a ScienceLogic Library

In addition to the ScienceLogic Libraries that are included in the latest version of Skylar One or included in a PowerPack, you can use the **ScienceLogic Library Manager** page to import additional libraries.

To import a library:

1. Go to the **ScienceLogic Library Manager** page (System > Customize > ScienceLogic Libraries).
2. Click the **[Actions]** menu and then select *Import ScienceLogic Library*.
3. The **Import ScienceLogic Library** dialog box appears. Use the **[Browse]** button to navigate to the ScienceLogic Library file you want to import and then click the **[Import]** button.
4. The imported ScienceLogic Library appears on the **ScienceLogic Library Manager** page.

Creating a ScienceLogic Library

You can create your own library to import into Skylar One in a py_package library file saved in uncompressed **.tar** format. This section describes the requirements for a ScienceLogic-compatible library.

Important Notes on Creating ScienceLogic Libraries

NOTE: Older versions of Skylar One bundled ScienceLogic Libraries in a py_directory format. That format has since been deprecated. Current ScienceLogic Libraries should be bundled only in a py_package format.

NOTE: The option to use Python 3.9 execution environments is limited to the Skylar One 12.2.1.1, 12.2.1.2, and 12.2.3 releases.

Additionally, the Skylar One 12.3.0 release removed support for Python 3.9 entirely and added support for Python 3.11, and Skylar One 12.5.20 added support for Python 3.12. Any Dynamic Applications that use Python 3.9 execution environments will stop working after upgrading to Skylar One 12.3.0 or later.

If you are currently using Python 3.9 execution environments, then after updating to 12.3.0 or later, you must create a Python 3.11 or later execution environment and align any Dynamic Applications that are currently aligned to the Python 3.9 execution environments to Python 3.11 or later execution environment to make them work again.

Python 2 support was deprecated in Skylar One 12.5.1.

The following are important items to note when creating a ScienceLogic library:

- Library files must be saved in uncompressed **.tar** format.
- For best results, use GNU tar rather than bsdtar.
- You can only include wheel files (.whl) as additional dependencies. You cannot include any additional packages in your library file that require compilation or linking because Python build tools are not included in the Skylar One platform.
- Any libraries that are bundled as dependencies in your py_package file will not be used if a newer version of that library exists on the appliance *unless* you have a restriction in your package specifying a given version to use.
- For example, if your library "my-library" requires the Python community package "requests" version 2.25.1, you would create the library "requests" version "2.25.1" first. You would then import the requests library into Skylar One and align it and "my-library" to your Execution Environment. At runtime, both libraries, at those specific versions, will be used.
- For an alternative example, if your library "my-library" requires the Python community package "requests", but the version is not important, you would include the "requests" package as a ".whl" file in your "my-library" library, adding it to the "manifest.txt" file. After aligning "my-library" to your Execution Environment, it will be available at runtime. The "requests" package included in the "manifest.txt" will be available during the install of "my-library", but may not be used if a newer acceptable package is found for install.
- For a py_package library, an optional "metadata.json" file can be included but it must include the requires_python element, otherwise an error will be displayed upon import.

TIP: A best practice is to version your libraries. Each time you make modifications to a library, create a new version and a new execution environment that uses that version. In that way, you can use different versions in different Dynamic Applications for testing purposes.

Library File Type: py_package

The py_package file type is the preferred library file type used to bundle Python packages and metadata files. A py_package must have the metadata files described below along with the .**whl** file(s), and it must be saved in uncompressed **.tar** format. You must create this tar file with Red Hat Enterprise Linux or a similar operating system to ensure compatibility. Tar files created on MacOS are not supported.

NOTE: All metadata files included in the package must be at the root level of the directory.

Alternately, you can include a "metadata.json" file instead of the following .**txt** files, but the "metadata.json" file must include the requires_python element, otherwise an error will be displayed upon import.

A py_package tar file must include the following files:

- *__init__.py*. This file is required for the directory to be used as a Python package. It identifies this as a package.
- *name.txt*. This text file specifies the name of the library and must match the name of the package being installed.
- *version.txt*. This text file specifies the version of the library and must match the version of the package being installed.
- *type.txt*. This text file specifies that the library is a py_package type library.
- *manifest.txt*. This text file includes the Python package name and version (as specified in the name.txt and version.txt files), followed by a list of additional supporting files that must be deployed to the internal package repository to use the library. For example, if the library is dependent on another package, that other package must be listed in the manifest file.

  NOTE: The ScienceLogic library and the Python package must have the same name, and this name must be the first entry in the manifest file.

- *Wheel (.whl) file*. This is a standard wheel file that you create. See Python documentation for more information about creating wheel files. Wheel files must be compatible with the version of Python used in the library. Wheel files must be able to work in Red Hat Enterprise Linux or a similar operating system, but they do not need to be created in Red Hat.

NOTE: ScienceLogic supports Python 3-compatible libraries and wheel files. Python 2 support was deprecated in Skylar One 12.5.1. However, if you are creating a Python 2 library in an older version of Skylar One prior to 12.5.1, then its corresponding wheel file must also be Python 2-compatible.

IMPORTANT: The `tar` command requires that you include all of the files, but not the current working directory. If you include the current working directory, then the import feature cannot find the root files. For the best results, use the following command:

```
tar cvf thing.tar *
```

instead of this command:

```
tar cvf thing.tar .
```

Manually Building a ScienceLogic Library

To manually build a ScienceLogic library, follow this example:

1. Use the sample python library called `silo_demo-0.0.7-py2.py3-none-any.whl`:
   - Create a project folder for your library, for example, `myproject`.
   - Copy your project's python wheel file into your new project folder.

   ```
   mkdir myproject
   ```

   ```
   cp silo_demo-0.0.7-py2.py3-none-any.whl myproject/
   ```

2. Download a complete set of python dependencies for your python package. In this example case, there is a dependency on `requests`. To download the dependency package, you need to change the project directory to download the package and any transitive dependencies using the Python binary path in your execution environment.

NOTE: Any non-pure Python dependencies included in your ScienceLogic Library must be built distributions (wheels) that do not require a build stage. Skylar One does not include compilation tools for source distributions.

```
cd myproject
```

```
python3 -m pip download --find-links=. silo_demo==0.0.7
```

NOTE: To build these packages, ScienceLogic recommends that you are on Oracle Linux or Redhat based x86-64 installation.

NOTE: Execute the `python3 -m pip download command` with a *python3.6* interpreter to operate correctly with version 11.3 or later of Skylar One. Some Linux distributions default the command python to *python2.7*.

- The corresponding request python package and all its dependencies will download into your project folder.

```
Looking in links: .

Processing ./silo_demo-0.0.7-py2.py3-none-any.whl

    File was already downloaded /ScienceLogic/myproject/silo_
    demo-0.0.7-py2.py3-none-any.whl

Collecting requests

    Using cached requests-2.29.0-py3-none-any.whl (62 kB)

Collecting certifi>=2017.4.17

    Using cached certifi-2022.12.7-py3-none-any.whl (155 kB)

Collecting urllib3<1.27,>=1.21.1

    Using cached urllib3-1.26.15-py2.py3-none-any.whl (140 kB)

Collecting charset-normalizer<4,>=2

    Using cached charset_normalizer-3.1.0-cp39-cp39-macosx_10_
    9_x86_64.whl (124 kB)

Collecting idna<4,>=2.5

    Using cached idna-3.4-py3-none-any.whl (61 kB)

Saved ./requests-2.29.0-py3-none-any.whl

Saved ./certifi-2022.12.7-py3-none-any.whl

Saved ./charset_normalizer-3.1.0-cp39-cp39-macosx_10_9_x86_
64.whl

Saved ./idna-3.4-py3-none-any.whl

Saved ./urllib3-1.26.15-py2.py3-none-any.whl

Successfully downloaded silo_demo requests certifi charset-
normalizer idna urllib3
```

- List the dependencies to ensure they are correct.

```
[myproject]$ ls -l
```

```
-rw-r--r--@ 1 dev staff 4524 May 2 11:31 silo_demo-0.0.7-
py2.py3-none-any.whl
```

```
-rw-r--r-- 1 dev staff 62499 May 3 16:31 requests-2.29.0-py3-
none-any.whl
```

```
-rw-r--r-- 1 dev staff 155255 May 3 16:31 certifi-2022.12.7-
py3-none-any.whl
```

```
-rw-r--r-- 1 dev staff 124711 May 3 16:31 charset_normalizer-
2.0.4-cp36-cp36mu-manylinux1_x86_64.whl
```

```
-rw-r--r-- 1 dev staff 61538 May 3 16:31 idna-3.4-py3-none-
any.whl
```

```
-rw-r--r-- 1 dev staff 140881 May 3 16:31 urllib3-1.26.15-
py2.py3-none-any.whl
```

CAUTION: Skylar One comes pre-installed with the proper versions of wheel, setuptools, and pip wheels for each release. Do not include these in your ScienceLogic library as it will cause a deployment failure.

3. Create the *metadata.json* file. Ensure that you are in your project folder and use your favorite editor tool to create the *metadata.json* file.

   The following example includes the minimum set of metadata required by Skylar One. The information supplied corresponds to the principle python library you set up in step 1. The environment deployer will `pip install` the principle python library based on the `name` and `version` metadata, so they must be accurate.

   Update the following fields with your information:

   - ***meta_version***. Leave this as default `"0.2"`.
   - ***description***. A short description of the ScienceLogic library.
   - ***manifest***. Contains the complete list of wheel files from Step 2.
   - ***name***. The exact name of the principle python package used by pip to install the python package. In this example, the name is `silo_demo`.
   - ***type***. Leave this as default `"py_package"`
   - ***version***. The precise version of the principle python package used by pip to install the python package. In this example, the version is `0.0.7`.

- ***requires_python***.This value specifies the versions of python that this ScienceLogic library is compatible with. Skylar One will use this value to filter the libraries that are compatible with the specified python version of the execution environment. For example, a ScienceLogic library with a setting of `"requires_python": ">=3.6"` would be compatible with any execution environment greater than or equal to 3.6. The string found in `"requires_python"` follows the same convention used for python packages. The **ScienceLogic Library Manager** page (System > Customize > ScienceLogic Libraries) sorts out the available libraries for a given execution environment based on the ***requires_python*** field.

CAUTION: The ***requires_python*** field is a crucial part of the ScienceLogic library metadata. An incorrect value in this field might lead to a situation where the execution environment will fail to deploy due to conflicts with the python version of the execution environment.

Code Example

```
{
    "meta_version": "0.2",
    "fields": {
            "description": "A demo ScienceLogic library",
            "manifest": ["certifi-2022.12.7-py3-none-any.whl",
            "charset_normalizer-2.0.4-cp36-cp36mu-manylinux1_x86_
            64.whl", "idna-3.4-py3-none-any.whl", "requests-2.29.0-
            py3-none-any.whl", "silo_demo-0.0.7-py2.py3-none-any.whl",
            "urllib3-1.26.15-py2.py3-none-any.whl"],
            "name": "silo_demo",
            "type": "py_package",
            "version": "0.0.7",
            "requires_python": ">=3.6"
        }
}
```

4. Validate the *metadata.json* file. Validating your JSON files ensures there aren't any errors in the pairing of braces and syntax. To validate your metadata.json file type:

```
python -m json.tool metadata.json
```

5. Create the ScienceLogic library archive.

Give your archive a unique name so that it is easy to recognize and distinguish from other library archives. Since it is possible to support multiple python runtime versions, ScienceLogic recommends that you include the python version(s) supported in the library archive you create.

Use the following naming convention as a guide:

```
<lib name> + <lib version>. <runtime version/s>.tar
```

NOTE: The following commands must be run from inside the directory you created in step 1. For the following examples, that is the `myproject` directory.

For example:

- Create the .tar archive:

```
tar cvf silo_demo+0.0.7.py36.tar *
```

- Confirm the creation of the .tar archive by listing List the .tar files in the directory:

```
ls -l *.tar
```

```
-rw-r--r-- 1 em7admin em7admin 480256 Nov 29 16:28 silo_
demo+0.0.7.py36.tar
```

- Check the contents of the .tar archive:

```
tar tvf silo_demo+0.0.7.py36.tar
```

```
-rw-r--r-- 1 em7admin em7admin 161120 Nov 29 16:12 certifi-2022.9.24-
py3-none-any.whl
```

```
-rw-r--r-- 1 em7admin em7admin 39748 Nov 29 16:12 charset_normalizer-
2.0.4-cp36-cp36mu-manylinux1_x86_64.whl
```

```
-rw-r--r-- 1 em7admin em7admin 61538 Nov 29 16:12 idna-3.4-py3-none-
any.whl
```

```
-rw-r--r-- 1 em7admin em7admin 479 Nov 29 16:25 metadata.json
```

```
-rw-r--r-- 1 em7admin em7admin 62843 Nov 29 16:12 requests-2.28.1-py3-
none-any.whl
```

```
-rw-r--r-- 1 em7admin em7admin 4524 Nov 29 16:12 silo_demo-0.0.7-
py2.py3-none-any.whl
```

```
-rw-r--r-- 1 em7admin em7admin 140572 Nov 29 16:12 urllib3-1.26.13-
py2.py3-none-any.whl
```

CAUTION: Depending on your development environment, your .tar archiving tool may generate an archive that is incompatible with the Linux distribution used by Skylar One. If this occurs, try extracting `( tar xvf myarchive.tar )` or just listing the archive `( tar tvf my archive.tar )` on your Skylar One appliance before archiving to ensure compatibility with the following steps.

CAUTION: Only the contents of the directory you created in step 1 should be listed when you execute the previous command. If the directory itself is included in the output, then the .tar archive will not work correctly when imported into Skylar One. If this occurs, see the following knowledge base article for more information, including steps to resolve the issue: https://support.sciencelogic.com/s/article/14578.

6. Import the ScienceLogic library. For instructions, see *Importing a ScienceLogic Library*.
7. Create a Skylar One Execution Environment.
   - To create your Python3 Execution Environment, go to the **Environment Manager** page (System > Customize > ScienceLogic Libraries > Actions > Execution Environments) and click **[New]**.
   - In the ***Environment Name*** field, select *python3*.
   - For this example, add the ***Environment Name*** `Snippet Framework Demo py36` and click **[Save]**. The compatible libraries are displayed and available to the python3 execution environment.
   - Click the lightning bolt (⚡) to include all of the libraries in this environment.

IMPORTANT: The string found in `requires_python` will follow the same convention used for python packages.

NOTE: A ScienceLogic library with `requires_python>=3.6` would be compatible with any execution environment greater than or equal to 3.6.

   - Make note of the environment GUID for the following steps. In this example, the environment GUID was:

```
0117E1C041D8CE0C69A8391C02E6DC3B
```

8. Align your ScienceLogic library to the Skylar One Execution Environment.

- Return to the **Environment Manager** page (System > Customize > ScienceLogic Libraries > Actions > Execution Environments). You should see that your ScienceLogic library is available for alignment to your execution environment.
- Click the lightning bolt ( ) to add your library to the environment.

9. Test your ScienceLogic library.
    - Log on to the collector and deploy the ScienceLogic Execution Environment.

```
sudo -u s-em7-core /opt/em7/bin/python3 /opt/em7/lib/python3/sl_envs/env_
manager.pyc --debug 0117E1C041D8CE0C69A8391C02E6DC3B
```

    - Activate your new environment.

```
source /opt/em7/envs/0117E1C041D8CE0C69A8391C02E6DC3B/bin/activate
```

    - From the python command line, try importing your new package and executing its API.

```
[uenv-3869008638788123352]$ python

Python 3.6.8 (default, Nov 18 2021, 10:07:16)

[GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> from silo.demo import demo_class

>>>
```

NOTE: ScienceLogic is working to minimize the development effort through the use of Snippet Framework and low code specifications. Unless your collection environment is unique, you may not need to develop content such as manually building your ScienceLogic library.

Converting a py_directory ScienceLogic Library

Older versions of Skylar One bundled ScienceLogic Libraries in a Python 2-compatible py_directory format. That format has since been deprecated. Current ScienceLogic Libraries should be bundled only in a Python 3-compatible py_package format.

To ensure your ScienceLogic Libraries continue operating as intended, you can convert Python 2 py_directory ScienceLogic Libraries to a wheel for use in Python 3-compatible py_package ScienceLogic Libraries.

Typically, a py_directory package consists of a few Python modules in a project folder that is only one level deep and consists only of Python content with no C extensions.

To perform this conversion, follow the example in this section, which uses a ScienceLogic Library called `clevon`. The original contents of this library looked like this:

```
tar tvf clevon_v_1.4_~=2.7.tar

drwxr-xr-x  0 user staff       0 Oct 18  2023 clevon/
-rw-r--r--  0 user staff   11341 Oct 18  2023 clevon/clevon_globals.py
-rw-r--r--  0 user staff    1017 Oct 18  2023 clevon/misc.py
-rw-r--r--  0 user staff    1425 Oct 18  2023 clevon/interface.py
-rw-r--r--  0 user staff       0 Oct 18  2023 clevon/__init__.py
-rw-r--r--  0 user staff    1975 Oct 18  2023 clevon/graphing.py
-rw-r--r--  0 user staff    2430 Oct 18  2023 clevon/have_children.py
-rw-r--r--  0 user staff    2399 Oct 18  2023 clevon/math_lesson.py
-rw-r--r--  0 user staff       4 Oct 17  2023 clevon/version.txt
-rw-r--r--  0 user staff      33 Oct 18  2023 clevon/description.txt
-rw-r--r--  0 user staff     463 Oct 18  2023 clevon/chores.py
```

1. To convert this format, you must create a new project folder (`project_folder`, for this example) with the following structure:

```
LICENSE.txt
README.md
pyproject.toml
src/
  clevon/
    clevon_globals.py
    misc.py
    interface.py
    __init__.py
    graphing.py
    have_children.py
    math_lesson.py
    chores.py
```

NOTE: The LICENSE.txt and README.md files can be empty. Also, in this example, the metadata version.txt and description.txt files were removed from the original py_ directory contents.

2. The contents of the `pyproject.toml` file in the above example should look similar to this:

```
[build-system]
requires = ["setuptools"]
build-backend = "setuptools.build_meta"

[project]
name = "clevon"
version = "1.6"
authors = [
  { name="ScienceLogic, Inc.", email="author@sciencelogic.com" },
]
description = "A Crucible test package"
readme = "README.md"
requires-python = ">=3.6"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]
```

3. Run the builder to package the wheel:

```
cd project_folder
```

```
python3 -m build
```

When the build finishes successfully, you should receive a message similar to this:

```
Successfully built clevon-1.6.tar.gz and clevon-1.6-py3-none-any.whl
```

NOTE: If the build does not work, you might need to first install the build package into your environment:

```
python3 -m pip install build
```

4. As part of the previous step, a `dist` directory should have been created that contains the built wheel. Confirm that the artifacts have been created:

```
ls -l dist
total 48
-rw-r--r--  1 dayn.harum  staff  9764 Sep  9 09:25 clevon-1.6-py3-
none-any.whl
-rw-r--r--  1 dayn.harum  staff  8618 Sep  9 09:25 clevon-1.6.tar.gz
```

5. Confirm that you can install, import, and uninstall the package:

```
cd dist
dist % python3 -m pip install --no-index --find-links=. clevon==1.6
Looking in links: .
Processing ./clevon-1.6-py3-none-any.whl
Installing collected packages: clevon
Successfully installed clevon-1.6

dist % python3
Python 3.9.5 (default, May 18 2021, 12:31:01)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> import clevon
>>> clevon.__path__
['/Users/user/miniconda3/lib/python3.9/site-packages/clevon']
>>> from clevon import graphing
>>>

python3 -m pip uninstall clevon==1.6
Found existing installation: clevon 1.6
Uninstalling clevon-1.6:
  Would remove:
    /Users/user/miniconda3/lib/python3.9/site-packages/clevon-
1.6.dist-info/*
    /Users/user/miniconda3/lib/python3.9/site-packages/clevon/*
Proceed (Y/n)? Y
  Successfully uninstalled clevon-1.6
```

6. After you have successfully built the wheel, you can follow the steps in the section *Manually Building a ScienceLogic Library* to include it in a ScienceLogic Library.

Deleting a ScienceLogic Library

The **ScienceLogic Library Manager** page allows you to delete ScienceLogic Libraries that are not currently in use.

Skylar One will not allow you to delete ScienceLogic Libraries that are aligned to execution environments that are being used by any Dynamic Applications, credential tests, or Run Book Actions. You also cannot delete libraries that are aligned to the "System" environment.

To delete a library:

1. Go to the **ScienceLogic Library Manager** page (System > Customize > ScienceLogic Libraries).
2. Select the ScienceLogic Library that you want to delete.
3. Click the ***Select Actions*** menu in the lower right of the page and select *DELETE ScienceLogic Library*, and then click the **[Go]** button.
4. The selected ScienceLogic Library is deleted from the **ScienceLogic Library Manager** page.

Chapter 3

Managing Execution Environments

Overview

Execution environments in Skylar One let you share code between snippets, Dynamic Applications, and PowerPacks. Execution environments have a global unique identifier (GUID), a name, and list of associated libraries. The libraries associated with an execution environment provide a way to bundle reusable code in a self-contained environment available in snippets. You can use different execution environments for different Dynamic Application requirements.

This chapter covers the following topics:

Viewing the List of Execution Environments 25

Creating an Execution Environment 25

Editing an Execution Environment 26

Copying an Execution Environment 27

Deleting an Execution Environment 27

Viewing the List of Execution Environments

The **Environment Manager** page (System > Customize > ScienceLogic Libraries > Actions > Execution Environments) displays a list of all existing execution environments.

For each execution environment, the page displays the following information:

TIP: To sort the list of execution environments, click on a column heading. The list will be sorted by the column value, in ascending order. To sort by descending order, click the column heading again.

- ***Environment Name***. The name of the execution environment.
- ***Env GUID***. The execution environment's globally unique identifier.
- ***Env Type***. The Python runtime version selected for the execution environment.
- ***Libs***. The number of ScienceLogic Libraries aligned with the execution environment. Click the View Aligned Libraries icon ( ) to view the list of ScienceLogic Libraries that are aligned to a particular execution environment.

Creating an Execution Environment

Skylar One includes several execution environments from preloaded PowerPacks and system updates for your use. If needed, you can also create new execution environments. After you have successfully created an execution environment, it will appear in the list of execution environments available for aligning with Dynamic Applications, credential tests, and Run Book Actions.

NOTE: The option to use Python 3.9 execution environments is limited to the Skylar One 12.2.1.1, 12.2.1.2, and 12.2.3 releases.

Additionally, the Skylar One 12.3.0 release removed support for Python 3.9 entirely and added support for Python 3.11, and Skylar One 12.5.20 added support for Python 3.12. Any Dynamic Applications that use Python 3.9 execution environments will stop working after upgrading to Skylar One 12.3.0 or later.

If you are currently using Python 3.9 execution environments, then after updating to 12.3.0 or later, you must create a Python 3.11 or later execution environment and align any Dynamic Applications that are currently aligned to the Python 3.9 execution environments to Python 3.11 or later execution environment to make them work again.

Python 2 support was deprecated in Skylar One 12.5.1.

To create an execution environment:

1. From the **Environment Manager** page (System > Customize > ScienceLogic Libraries > Actions > Execution Environments), click the **[New]** button.

2. On the **Environment Editor** page, type a name for the new execution environment in the ***Environment Name*** field, select a Python version from the ***Environment Type*** drop-down, and then click the save icon ( ).

> **NOTE:** Each execution environment is assigned an Env GUID, which makes it unique. Since two environments can share a name, it is a best practice to use the Name field to version an environment when updating its contents.

3. The **Library Alignment** pane appears, displaying a list of ScienceLogic Libraries that you can align with the new execution environment. Click the align icon ( ) for a ScienceLogic Library to align that library to the new execution environment.

> **TIP:** You can select the ***Filter Libraries By Python Version*** drop-down to filter ScienceLogic libraries compatible with the execution environment runtime. This feature is set to ***on*** by default.

4. Align any additional ScienceLogic libraries to the execution environment as needed.

> **TIP:** If necessary, you can click the unalign icon ( ) for a ScienceLogic Library to remove that library's alignment with the new execution environment.

5. When you are finished, click the save icon ( ) again to save the execution environment with the ScienceLogic Library alignment you selected, and then click **[Close]**.

Editing an Execution Environment

You can edit an execution environment by clicking its wrench icon ( ). When you do so, you can change the environment's name, align additional ScienceLogic Libraries to the environment, or remove its alignment with one or more libraries.

> **WARNING:** Do not edit ScienceLogic-created execution environments. You may make a copy of the execution environment, but ScienceLogic often updates these execution environments in a release, which will overwrite any existing execution environment of the same globally unique identifier (GUID).

To edit an execution Environment:

1. From the **Environment Manager** page (System > Customize > ScienceLogic Libraries > Actions > Execution Environments), click the edit icon ( ) for the execution environment that you want to edit.

2. On the **Environment Editor** page, you can do any of the following:
    - Change the environment's name in the ***Environment Name*** field.
    - Align additional ScienceLogic Libraries to the execution environment by clicking the appropriate align icons ( ).
    - Remove the execution environment's alignment with one or more libraries by clicking the appropriate unalign icons ( ).
3. When you are finished, click the save icon ( ) and then click **[Close]**.

Copying an Execution Environment

If you want to create a new execution environment that has traits similar to an existing environment, you can make a copy of the existing execution environment that you can then edit as needed. When you copy an existing execution environment, the new environment includes the same libraries as the original environment, but it will have a new globally unique identifier (GUID).

> CAUTION: Making a copy of an execution environment does not mean there will be a duplicate runtime environment. Environments that share a set of libraries will result in the same virtual runtime environment to optimize space on the Data Collector.

To copy an execution environment:

1. From the **Environment Manager** page (System > Customize > ScienceLogic Libraries > Actions > Execution Environments), click the copy icon ( ) for the execution environment that you want to copy. A confirmation message appears.
2. Click **[OK]** to continue. A copy of the selected execution environment appears on the **Environment Manager** page. By default, it has the same name as the original execution environment, followed by the word "(copy)".

Deleting an Execution Environment

If you no longer need an execution environment and it is not currently aligned with a Dynamic Application, credential test, or Run Book Action, then you can delete it. Skylar One will not allow you to delete execution environments that are being used by any Dynamic Applications, credential tests, or Run Book Actions.

> CAUTION: Do not delete execution environments that are shipped with Skylar One.

To delete an execution environment:

1. From the **Environment Manager** page (System > Customize > ScienceLogic Libraries > Actions > Execution Environments), click the delete icon ( ) for the execution environment that you want to delete. A confirmation message appears.
2. Click **[OK]** to continue. The execution environment is deleted from the **Environment Manager** page.

© 2003 - 2026, ScienceLogic, Inc.

All rights reserved.

ScienceLogic™, the ScienceLogic logo, and ScienceLogic's product and service names are trademarks or service marks of ScienceLogic, Inc. and its affiliates. Use of ScienceLogic's trademarks or service marks without permission is prohibited.

ALL INFORMATION AVAILABLE IN THIS GUIDE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED. SCIENCELOGIC™ AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT.

Although ScienceLogic™ has attempted to provide accurate information herein, the information provided in this document may contain inadvertent technical inaccuracies or typographical errors, and ScienceLogic™ assumes no responsibility for the accuracy of the information. Information may be changed or updated without notice. ScienceLogic™ may also make improvements and / or changes in the products or services described herein at any time without notice.

800-SCI-LOGIC (1-800-724-5644)

International: +1-703-354-1010