



Integration Service for Developers

Version 2.0.0

Table of Contents

Introduction to the Integration Service Tools	4
What is the Integration Service?	5
Tools Included with Integration Service	6
Prerequisites for Creating Integration Applications	7
Creating a Step	8
What is a Step?	9
Default Steps	10
Requirements	10
Creating a Step from the Step Template	11
Including the SubClass and Required Methods	11
Subclass	11
Required Methods	12
Defining the Logic for the Step	12
The init Method	12
The execute Method	13
Transferring Data Between Steps	14
Saving Data for the Next Step	14
Retrieving Data from a Previous Step	14
get_data_from_step_by_name	15
get_data_from_step_by_order	15
join_previous_step_data	16
Step Parameters	17
Base Parameters Available in All Steps	17
Defining a Parameter	17
Retrieving Parameter Values	18
Variable Substitution in Parameters	18
Retrieving Variable Values	19
Defining Logging for the Step	20
Raising Exceptions	20
Uploading Your Step	20
Uploading a Step with iscli	21
Uploading a Step with the API	21
Validating Your Step	22
Viewing Logs	22
Creating an Integration Application	25
What is an Integration Application?	26
Default Integration Application	27
Requirements	27
Creating an Integration Application from the Integration Template	27
Defining Required Fields for the Integration Application	28
Creating the List of Steps and Step Parameters	29
Specifying Values for name and file	30
Specifying the Parameters	30
Transferring Data Between Steps	30
Defining Retry Options for a Step	32
Defining Variables for an Integration Application	33
Uploading the Integration Application to the Integration Service System	34
Uploading an Integration Application with iscli	35
Uploading an Integration Application with the API	35
Running the Integration Application	36

Running the Integration Application with a Custom Queue	36
Defining a Custom Queue	37
Configuring an Integration Application to Use a Custom Queue	39
Configuring an Integration Application to Always Use a Custom Queue.	39
Configuring an Integration Application to Use a Custom Queue at Run Time	40
Defining a Configuration	40
Uploading the Configuration File to the Integration Service System	42
Uploading a Configuration with iscli	42
Uploading a Configuration with the API	43
Viewing a Report for an Integration Application	44
Viewing Logs for an Integration Application	46
Working with Integration Applications in the Integration Service User Interface	47
Editing an Integration Application in the User Interface	48
Configuring an Integration Application in the User Interface	49
Running or Stopping an Integration Application in the User Interface	51
Creating a Synchronization PowerPack	53
What is a Synchronization PowerPack?	54
Generating the Synchronization PowerPack Structure	54
Synchronization PowerPack Structure	55
Synchronization PowerPack Properties	55
Building a Synchronization PowerPack	56
Building a Synchronization PowerPack with Python3	56
Building a Synchronization PowerPack with the iscli Utility	56
Uploading a Synchronization PowerPack to the Integration Service	57
Uploading a Synchronization PowerPack with the User Interface	57
Uploading a Synchronization PowerPack with the iscli Utility	57
Activating and Installing a Synchronization PowerPack	58
Activating and Installing with the User Interface	58
Activating and Installing with the iscli Utility	58
Extending an Existing Synchronization PowerPack	59
Restrictions	59
Modifying Integration Application Definitions	59
Modifying a Step	59
Full Extended Step Example	60
ipaa.score.BaseStep class	62
get_app_variable	63
Description	63
Syntax	63
Parameters	63
Return	63
Example	63
get_available_previous_step_input_positions	64
Description	64
Syntax	64
Return	64
get_data_from_step_by_name	65
Description	65
Syntax	65
Parameters	65
Return	65
Example	65
get_data_from_step_by_order	66

Description	66
Syntax	66
Parameters	66
Return	66
Exception	66
Example	66
get_name	67
Description	67
Syntax	67
Return	67
get_parameter	68
Description	68
Syntax	68
Parameters	68
Return	68
Example	68
get_parameter_from_previous_step	69
Description	69
Syntax	69
Parameters	69
Return	69
join_previous_step_data	70
Description	70
Syntax	70
Parameters	70
Return	70
Example	71
new_step_parameter	72
Description	72
Syntax	72
Parameters	72
Example	72
save_data_for_next_step	73
Description	73
Syntax	73
Parameters	73
Example	73
validate_parameter_values	74
Description	74
Syntax	74
Tools for the Integration Service	75
iscli	76
Syntax	76
List of Arguments	76
Example	78
IS API	78
Viewing API Documentation	78
Available Endpoints	78
POST	78
Querying for the State of an Integration Application	79
GET	79
REST	80

Chapter

1

Introduction to the Integration Service Tools

Overview

This manual describes how you can use the tools included in the Integration Service and your own tools to create your own integration applications, steps, and configurations that you can use with the Integration Service.

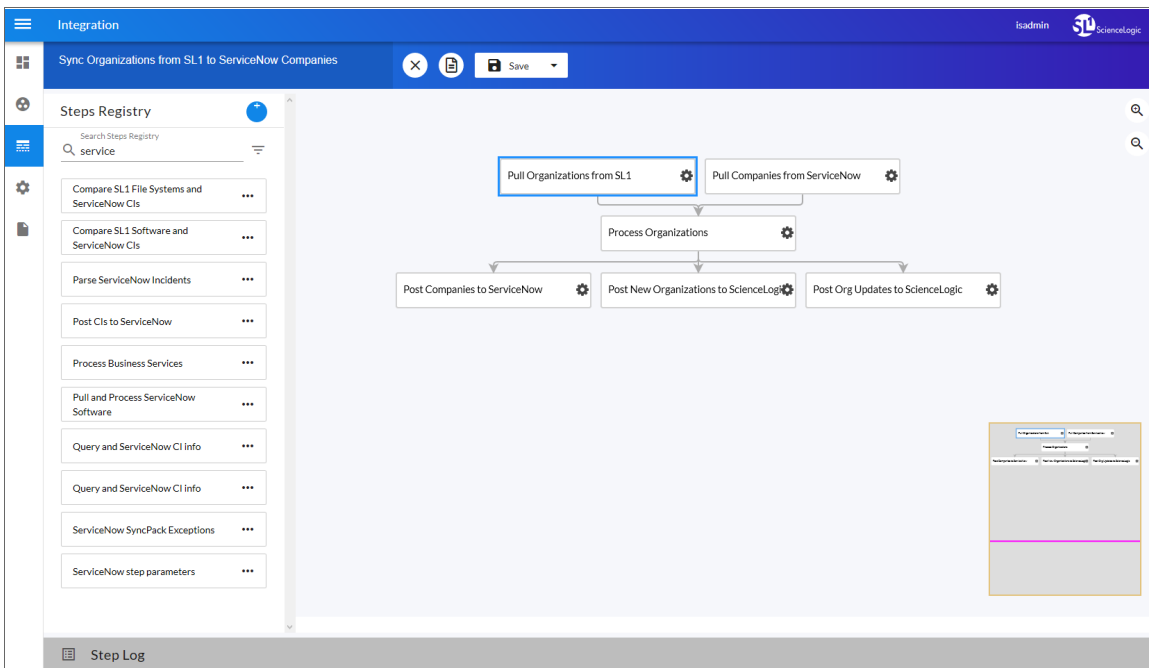
This chapter covers the following topics:

<i>What is the Integration Service?</i>	5
<i>Tools Included with Integration Service</i>	6
<i>Prerequisites for Creating Integration Applications</i>	7

What is the Integration Service?


The Integration Service enables intelligent, bi-directional communication between the ScienceLogic data platform and external data platforms to promote a unified management ecosystem. The Integration Service allows users to translate and share data between SL1 and other platforms without the need for programming knowledge. The Integration Service is designed to provide high availability and scalability.



The following image shows an example of an integration application and its steps in the Integration Service user interface:



The key elements of the Integration Service user interface include the following:

- **Steps.** A step is a generic Python class that performs a single action. Steps can accept zero or many input parameters or data from previous steps, and steps can specify output to be used by other steps. The input parameters are configurable variables and values used during execution. Steps can be re-used in multiple integrations applications. When these steps are combined in an integration application, they provide a workflow that satisfies a business requirement. All Python step code should be Python 3.7 or later. In the image above, the steps display as part of the flowchart in the main viewing pane as well as the **Steps Registry** pane.
- **Integration Applications.** An integration application is a JSON object that includes all the information required for executing an integration on the Integration Service platform. An integration application combines a set of steps that execute a workflow. The input parameters for each step are also defined in the application and can be provided either directly in the step or in the parent integration application. In the image above, the group of connected steps in the large pane make up the "Sync Organizations from SL1 to ServiceNow

Companies' integration application. You can access all integration applications on the **Integrations** page ().

- **Configuration Objects.** A configuration object is a stand-alone JSON file that contains a set of configuration variables used as input for an application. Configurations can include variables like hostname, user name, password, or other credential information. Configuration objects allow the same application to be deployed in multiple Integration Service instances, with different configurations. Click the **[Configure]** button from an integration application in the Integration Service user interface to access the configuration object for that integration application. You can access all configuration objects on the **Configurations** page ().
- **Synchronization PowerPack.** A Synchronization PowerPack (also called a SyncPack) contains all the code and logic needed to perform integrations on the Integration Service platform. You can access the latest steps, integration applications, and configurations for the Integration Service or a third-party integration (such as ServiceNow) by downloading the most recent Synchronization PowerPack for that integration from ScienceLogic. You can access all Synchronization PowerPacks on the **SyncPacks** page ().

The topics in this section describe how to create a step that is compatible with the Integration Service system. These topics also describe parameters, built-in parameters, and transferring data.

After you have created one or more steps, you can use them in one or more [integration applications](#).

Tools Included with Integration Service

The Integration Service includes the following tools for creating custom integrations:

- **iscli.** The Integration Service includes a command line tool called **iscli** (Integration Service Command Line Interface). When you install the Integration Service, **iscli** is automatically installed. The **iscli** allows you to upload integration applications, steps, and configurations. For more information on the **iscli**, see the section on [iscli](#).
- **IS API.** The Integration Service includes an API. When you install the Integration Service, the API is available. For more information on the IS API, see the section on [the IS API](#).
- **ipaascore.BaseStep class.** The Integration Service includes a Python class called **ipaascore.BaseStep class**. This class includes multiple pre-defined functions that you can use when you are writing or editing a step. For more information, see the section on [ipaascore.BaseStep class](#).
- **Base Steps Synchronization PowerPack.** The Base Steps Synchronization PowerPack includes a number of basic steps that perform common API calls. For more information, see [Default Steps](#).

Prerequisites for Creating Integration Applications

To create your own integration application, you must:

- Deploy an Integration Service system and ensure it is accessible. For details, see the *Integration Service Platform* manual.
- Have SSH or console access to the Integration Service system, so you can use the iscli (Integration Service Command Line Interface). You will use the iscli to upload steps, integration applications, and configurations to the Integration Service.
- Be comfortable with Python.
- Install a local copy of a Python IDE to use for development.
- Install a local copy of an API tool, like cURL, Insomnia, or Postman.
- Install a local copy of a source-code editor like Notepad++, vi, or TextEdit.

Chapter

2

Creating a Step

Overview

This chapter explains how to create your own custom step that you can then upload and run on an Integration Service system.

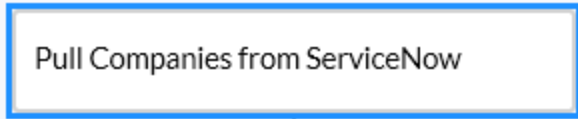
NOTE: All Python step code should be Python 3.7 or later.

This chapter covers the following topics:



<i>What is a Step?</i>	9
<i>Default Steps</i>	10
<i>Requirements</i>	10
<i>Creating a Step from the Step Template</i>	11
<i>Including the SubClass and Required Methods</i>	11
<i>Defining the Logic for the Step</i>	12
<i>Transferring Data Between Steps</i>	14
<i>Step Parameters</i>	17
<i>Defining Logging for the Step</i>	20
<i>Raising Exceptions</i>	20
<i>Uploading Your Step</i>	20
<i>Validating Your Step</i>	22
<i>Viewing Logs</i>	22

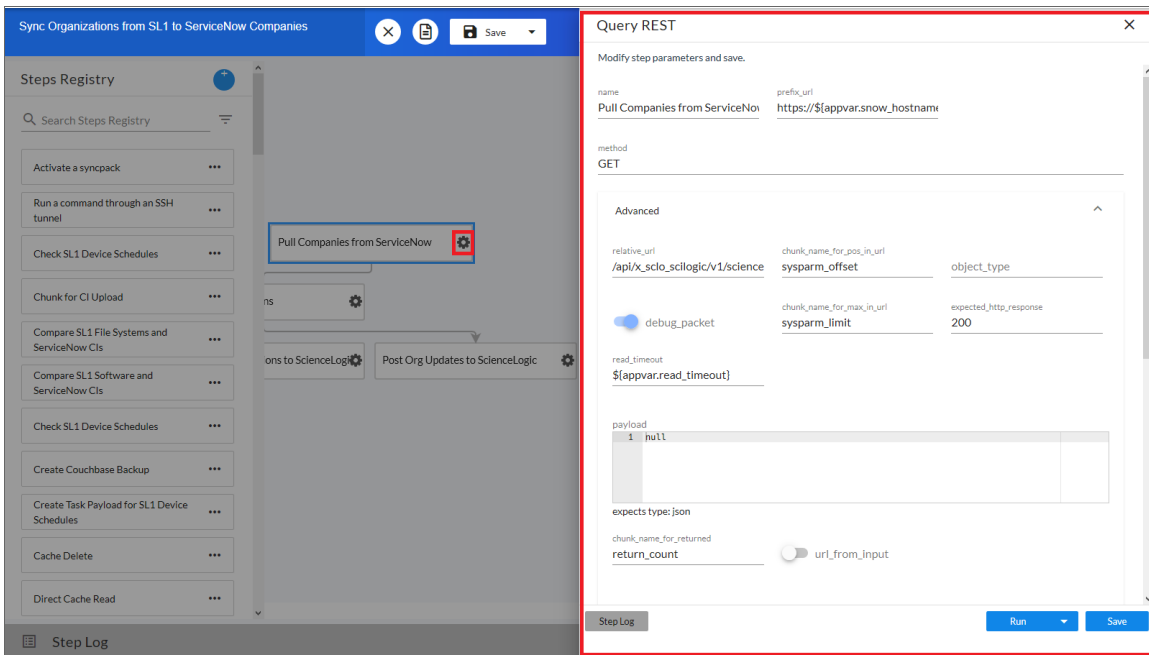
What is a Step?

In an Integration Service system, a **step** is a generic Python class that performs a single action, such as caching device data:



Steps accept arguments called **input parameters**. The parameters specify the values, variables, and configurations to use when executing the step. Parameters allow steps to accept arguments and allow steps to be re-used in multiple integrations. For example, you can use the same step to query both the local system and another remote system; only the arguments, such as hostname, username, and password change.

You can view and edit the parameters for a step by opening the integration application, clicking **[Open Editor]** (), and then clicking the gear icon () on a step to open the **Step Editor** pane:



A step can pass the data it generates during execution to a subsequent step. A step can use the data generated by another step. Also, you can add test data to the step and click **Custom Run** to run test data for that step.

The Integration Service system analyzes the required parameters for each step and alerts you if any required parameters are missing before the Integration Service runs the step.

Steps are grouped into the following types:

- **Standard**. Standard steps do not require any previously collected data to perform. Standard steps are generally used to generate data to perform a transformation or a database insert. These steps can be run independently and concurrently.
- **Aggregated**. Aggregated steps require data that was generated by a previously run step. Aggregated steps are not executed by the Integration Service until all data required for the aggregation is available. These steps can be run independently and concurrently.
- **Trigger**. Trigger steps are used to trigger other integration applications. These steps can be configured to be blocking or not.

Default Steps

The *Base Steps Synchronization PowerPack* version 1.0.0 contains the "Template App" integration application. You can use the "Template App" integration application as a template for building new integration applications.

The *Base Steps Synchronization PowerPack* also includes the following steps, which you can use in new and existing integration applications:

- Cache Delete
- Cache Read
- Cache Save
- Direct Cache Read
- MS-SQL Describe
- MS-SQL Insert
- MS-SQL Select
- MySQL Describe
- MySQL Insert
- MySQL Select
- Query GraphQL
- Query REST
- Run a command through an SSH tunnel
- Trigger Application

TIP: To view the code for a step, select a Synchronization PowerPack from the **SyncPacks** page, click the **[Steps]** tab, and select the step you want to view.

Requirements

To create a custom step, you must perform the following tasks:

1. Download or copy the step template, called *stepTemplate*.
2. Set up the required classes and methods in the step.
3. Define logic for the step, including transferring data between steps.
4. Define parameters for the step.
5. Define logging for the step.
6. Define exceptions for the step.
7. Upload the step to the Integration Service system.
8. Validate and test the step.

Creating a Step from the Step Template

The easiest way to create a new step is to use the step template that is included with the Integration Service system. To copy this template to your desktop:

1. Using an API tool like Postman or cURL, use the API [GET /steps/{step_name}](#):

```
GET URL_for_your_Integration_Service_system/api/v1/steps/stepTemplate
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.

2. Select and copy all the text in the *stepTemplate*.
3. Open a source-code editor and paste the content of the *stepTemplate* in the source-code editor.
4. Save the new file as *newfilename.py*

where:

- *newfilename.py* is the new name of the step and includes the *.py* suffix. The file name must be unique within your Integration Server system and cannot contain spaces. Note that the step name will also be the name of the Python class for the step.

Including the SubClass and Required Methods

To execute successfully on the Integration Service system, your step must be a subclass of the `ipaa.score.BaseStep` class and include the `init` method and the `execute` method.

Subclass

The *stepTemplate.py* file is already configured to include the new step as a subclass of the `ipaa.score.BaseStep` class. To update your step:

1. Use a source-code editor to open the re-named file for editing.
2. Notice that the file includes these lines of text:

```
from ipaascore.BaseStep import BaseStep
from ipaascommon import ipaas_exceptions
```

3. Do not remove or alter these lines of text.
4. Search for the following:

```
class stepTemplate(BaseStep):
```

5. Replace **stepTemplate** with the new name of the file (without the .py suffix).
6. Save and close the file.

Required Methods

To execute successfully on the Integration Service system, your step must contain at least two methods:

- **init method**. Allows you to define initialization options and parameters for the step.
- **execute method**. The execute method includes the logic for the step and performs the action. After the Integration Service system evaluates all parameters and initialization settings and aligns the step with a worker process, the Integration Service system examines the execute method.

Without these methods, the Integration Service system will consider your step to be "incomplete" and will not execute the step.

The stepTemplate.py file includes these two methods and the syntax of some of the sub-methods you can use within the main methods.

Defining the Logic for the Step

Each step requires the init method and the execute method. Within those methods, you can specify parameters and logic for the step. The following sections describe how to do this.

The init Method

From the init method, you can define the friendly name, the step description, and the step version:

```
self.friendly_name = "friendly name of the step. This name appears in the user
interface"
self.description = "Description of the step"
self.version = "version number"
```

In the QueryREST step, the friendly name, description, and version number are defined like this:

```
def __init__(self):
    self.friendly_name = "Query REST"
    self.description = "Step facilitates REST interactions and will return the
returned data dictionary and specified headers as data to the next step"
    self.version = "1.0.0"
```

From the init method, you also define the parameters for the step. The Integration Service system will examine the parameters and enforce the parameters when the step is run.

For example, if you specify a parameter as required, and the user does not specify the required parameter when calling the step, the Integration Service system will display an error message and will not execute the step.

To define a parameter, use the self.[new_step_parameter](#) function.

```
self.new_step_parameter(name=parameter_name, description="description", sample_value="sample value", default_value=None, required=False)
```

where:

- *name*. The name of the parameter. This value will be used to create a name:value tuple in the integration application file (in JSON).
- *description*. A description of the step parameter.
- *sample_value*. A sample value of the required data type or schema.
- *default_value*. If no value is specified for this parameter, use the default value. Can be any Python data structure. To prevent a default value, specify "None".
- *required*. Specifies whether this parameter is required by the step. The possible values are "True" or "False".

Here is an example from the QueryREST step (available on each Integration Service system):

```
self.new_step_parameter(name=PREFIX_URL, description="used with relative_url to create the full URL.", sample_value="http://10.2.11.253", default_value=None, required=True)
```

The execute Method

From the execute method you can:

- Use Python logic and functions
- retrieve the value of a parameter with self.[get_parameter](#)
- retrieve the value of a variable with self.[get_app_variables](#) (variables are defined in the integration application)
- [Retrieve data from a previous step](#) in the integration application
- [Save data for use by the next step](#) in the integration application
- Define logging for the step
- Define exceptions for the step

For details on all the functions you can use in the execute method, see the chapter on the [ipaa.score.BaseStep class](#).

You can also define additional methods in the step. For examples of this, see the QueryREST step provided with the Integration Service system.

For examples of the logic in a step, view one or more of the steps listed in [Default Steps](#).

Transferring Data Between Steps

An essential part of integrations is passing data between tasks. The Integration Service system includes native support for saving and transferring Python objects between steps. The [ipaascore.BaseStep class](#) includes multiple functions for transferring data between steps.

Saving Data for the Next Step

The [save_data_for_next_step](#) function saves an object or other type of data and make the data available to another step. The object to be saved and made available must be able to be serialized with pickle:

```
save_data_for_next_step(data_to_save)
```

where:

- `data_to_save` is a variable that contains the data.

NOTE: The `data_to_save` object must be of a data type that can be pickled by Python: None, True and False, integers, long integers, floating point numbers, complex numbers, normal strings, unicode strings, tuples, lists, set, and dictionaries.

The following is an example of the [save_data_for_next_step](#) function:

```
save_data = {'key': 'value'}  
self.save_data_for_next_step(save_data)
```

The integration application must then specify that the data from the current step should be passed to one or more subsequent step, using the **output_to** parameter. For details, see the section about [using integration applications to transfer data between steps](#).

Retrieving Data from a Previous Step

The [ipaascore.BaseStep class](#) includes multiple functions that retrieve data from a previous step:

- [get_data_from_step_by_name](#)
- [get_data_from_step_by_order](#)
- [join_previous_step_data](#)

NOTE: To retrieve data from a previous step, that previous step must save the data with the [save_data_for_next_step](#) function, and the integration application must specify that the data from the previous step should be passed to the current step using the **output_to** parameter.

get_data_from_step_by_name

The [get_data_from_step_by_name](#) function retrieves data saved by a previous step.

NOTE: Although the [get_data_from_step_by_name](#) function is simple to use, it does not allow you to write a generic, reusable step, because the step name will be hard-coded in the function. The functions [join_previous_step_data](#) or [get_data_from_step_by_order](#) allow you to create a more generic, reusable step.

```
get_data_from_step_by_name(step_name)
```

where:

- *step_name* is the name of a previous step in the integration application.

The following is an example of the [get_data_from_step_by_name](#) function:

```
em7_data = self.get_data_from_step_by_name('FetchDevicesFromEM7')
snow_data = self.get_data_from_step_by_name('FetchDevicesFromSnow')
```

get_data_from_step_by_order

The [get_data_from_step_by_order](#) function retrieves data from a step based on the position of the step in the integration application.

```
get_data_from_step_by_order(position)
```

where:

- *position* is the position of the step (the order that the step was run) in the integration application. Position starts at "0" (zero).

For example:

- Suppose your integration application has four steps: stepA, stepB, stepC, and stepD
- Suppose stepA was run first (position "0") and includes the parameter "output_to":["stepD"]
- Suppose stepB was run second (position "1") and includes the parameter "output_to":["stepD"]
- Suppose stepC was run third (position "2") and includes the parameter "output_to":["stepD"]
- Suppose stepD was run fourth
- If the current step is stepD, and stepD needs the data from stepC, you could use the following:

```
data_from_stepC = self.get_data_from_step_by_order(2)
```

join_previous_step_data

The `join_previous_step_data` function is the easiest and most generic way of retrieving data from one or more previous steps in the integration application.

If you are expecting similar data from multiple steps, or expecting data from only a single step, the `join_previous_step_data` function is the best choice.

The `join_previous_step_data` function gathers all data from all steps that included the `save_data_for_next_step` function and also include the `output_to` parameter in the integration application. By default, this function returns the joined set of all data that is passed to the current step. You can also specify a list of previous steps from which to join data.

The retrieved data must be of the same type. The data is then combined into a list in a dictionary.

If the data types are not the same, then the function will raise an exception.

```
join_previous_step_data([step_name])
```

where:

- `step_name` is an optional argument that specifies the steps. For example, if you wanted to join only the data from stepA and stepD, you could specify

```
self.join_previous_step_data(["stepA", "stepD"]),
```

The following is an example of the `join_previous_step_data` function in the QueryREST step (included in each Integration Service system):

```
def query_with_url_generated_from_input(self):
    """
    Iterates over data from previous steps and generates a relative url for each. Then
    executes that command
    :return:
    """
    count = 0
    input_data = self.join_previous_step_data()
    payload = self.get_parameter(PAYLOAD)
    if type(input_data) is list:
        for input_d in input_data:
            relative_url = self.get_parameter(RELATIVE_URL, input_d)
            self.process_REST_command(payload, relative_url)
            count += 1
    elif type(input_data) is dict:
        relative_url = self.get_parameter(RELATIVE_URL, input_data)
        self.process_REST_command(payload, relative_url)
        count += 1
    else:
        raise NotImplementedError("Data type: {} is not currently supported for generating
        relative urls form data".format(type(input_data)))
```

Step Parameters

Steps accept arguments, called **parameters**. These arguments specify the values, variables, and configurations to use when executing.

Base Parameters Available in All Steps

The Integration services BaseClass has a few base parameters that are automatically inherited by all steps and cannot be overwritten. You do not need to define these parameters before using them in steps:

- **name**. The Application-unique name for this step. That parameter can be used by other steps to refer to a step.
- **file**. The name of the file that will be executed by the step. For example, you could write step logic in a single file, but use that step logic with different integration applications and use different names for the step in each integration application.
- **output_to**. A list indicating that the data retrieved from this step should be output to another step. Setting this parameter links the steps, and the subsequent step will be able to retrieve data from the current step. The format is:

```
"output_to":["stepA", "stepB"]
```

Defining a Parameter

From the init method, you define the parameters for the step. The Integration Service system will examine the parameters and enforce the parameters when the step is run.

For example, if you specify a parameter as required, and the user does not specify the required parameter when calling the step, the Integration Service system will display an error message and will not execute the step.

To define a parameter, use the [new_step_parameter](#) function.

```
new_step_parameter(name='', description='', sample_value='', default_value=None,  
required=False)
```

where:

- **name**. The name of the parameter. This value will be used to create a name:value tuple in the integration application file (in JSON).
- **description**. A description of the step parameter.
- **sample_value**. A sample value of the required data type or schema.
- **default_value**. If no value is specified for this parameter, use the default value. Can be any combination of alphanumeric characters. To prevent a default value, specify "None".
- **required**. Specifies whether this parameter is required by the step. The possible values are "True" or "False".

Here is an example from the QueryREST step:

```
self.new_step_parameter(name=PREFIX_URL, description="used with relative_url to
create the full URL.", sample_value="http://10.2.11.253", default_value=None,
required=True)
```

Retrieving Parameter Values

To retrieve the latest value of a parameter, use the `get_parameter` function.

```
get_parameter(param_name, lookup_data=None)
```

where:

- `param_name` is the name of the parameter that you want to retrieve the value for.
- `lookup_data` is an optional dictionary that can provide a reference for additional variable substitutions.

For example, suppose we defined this parameter in the step named "GETgoogle.com":

```
self.new_step_parameter(name=prefix_url, description="used with relative_url to create
the full URL.", sample_value="http://10.2.11.253", default_value=None, required=True)
```

Suppose in the integration application that calls "GETgoogle.com", we specified:

```
"steps": [
  {
    "file": "QueryREST",
    "method": "GET",
    "name": "GETgoogle.com",
    "output_to": ["next_step"],
    "prefix_url": "http://google.com"
  }
],
```

Suppose we use the `get_parameter` function in the step "GETgoogle.com" to retrieve the value of the "prefix_url" parameter:

```
build_url_1 = self.get_parameter("prefix_url")
```

The value of `build_url_1` would be "http://google.com".

Variable Substitution in Parameters

The Integration Service system allows users to define variables, so that parameters can be populated dynamically.

To include a variable in a parameter, use the following syntax:

```
${exampleVariable}
```

The Integration Service system includes the following types of variables for use in parameters:

- `${object_from_previous_step}`. The Integration Service system will search the data from the previous steps for `object_from_previous_step`. If found, the Integration Service system will substitute the value of the object for the variable
- `${config.exampleVariable}`. Configuration variables are defined in a stand-alone file that lives on the Integration Service system and can be accessed by all integration applications and their steps. Including the **config.** prefix with a variable tells the Integration Service system to look in a configuration file to resolve the variable. If you want to re-use the same settings (like hostname and credentials) between applications, define configuration variables.
- `${appvar.exampleVariable}`. Application variables are defined in the integration application. These variables can be accessed only by steps in the integration application. Including the **appvar.** prefix with a variable tells the Integration Service system to look in the integration application to resolve the variable.
- `${stepfunc.exampleFunctionargs}`. The variable value will be the output from the user-defined function, specified in `exampleFunction`, with the arguments specified in `args`. The `exampleFunction` must exist in the current step. Additional parameters can be specified as `args` with a space delimiter. You can also specify additional variable substitution values as the arguments. This allows you to dynamically set the value of a variable using a proprietary function, with dynamically generated arguments. For example:

```
"param": "${step_func.add_numbers 1 2}"
```

will call a function (defined in the current step) called "add_numbers" and pass it the arguments "1" and "2". The value of "param" will be "3".

For details on defining configuration variables and application variables, see [the chapter on defining an integration](#).

Retrieving Variable Values

You can use the [get_app_variable](#) function to retrieve the latest value of a variable.

```
get_app_variable (variable_name)
```

where:

- `variable_name` is the name of the application variable that you want to retrieve the value for.

For example, suppose we defined this application variable in the integration application:

```
"app_variables": [  
  {  
    "name": "s11_hostname",  
    "description": "The SL1 hostname to participate in the sync",  
    "sample_value": "10.2.253.115",  
    "default_value": null,  
    "required": true,  
    "value": 10.64.68.25  
  },  
]
```

Suppose this integration application calls the step "sync_SL1_data".

In the step "sync_SL1_data", we could use the following function to resolve the value of "sl1_hostname":

```
hostname = self.get_app_variable("sl1_hostname")
```

The value of **hostname** would be "10.64.68.25".

Defining Logging for the Step

The Integration Service system includes a logger for steps. The BaseStep class initializes the logger, so it is ready for use by each step.

To define logging in a step, use the following syntax:

```
self.logger.logging_level ("log_message")
```

where:

- *logging_level* is one of the following Python logging levels:
 - critical
 - error
 - info
 - warning
- *log_message* is the message that will appear in the step log.

For example :

```
self.logger.info("informational message")
```

Raising Exceptions

The Integration Service platform natively handles exceptions raised from custom steps. You can include a user-defined exception or any standard Python exception.

If an exception is raised at runtime, the step will immediately be marked as a failure and be discarded.

To view the exception and the complete stack trace, use the steps from the section on [Viewing Logs](#).

Uploading Your Step

When you create a new step or edit an existing step, you must upload the step to the Integration Service system.

There are two ways to upload a step to the Integration Service system:

- At the command line with the `iscli` utility
- With the Integration Service API

Uploading a Step with `iscli`

The Integration Service system includes a command line tool called **`iscli`**. When you install Integration Service system, **`iscli`** is automatically installed.

To upload a step to the Integration Service system using `iscli`:

1. Either go to the console of the Integration Service system or use SSH to access the server.
2. Log in as **`isadmin`** with the appropriate password.
3. Enter the following at the command line:

```
iscli -u -s -f path_and_name_of_step_file.py -H hostname_or_IP_address_of_integration_service_system -P port_number_of_http_on_integration_service_system -u user_name -p password
```

where:

- *name_of_step_file* is the full pathname for the step.
- *hostname_or_IP_address_of_integration_service_system* is the hostname or IP address of the Integration Service system.
- *port_number_of_http_on_integration_service_system* is the port number to access the Integration Service system. The default value is 443.
- *user_name* is the user name you use to log in to the Integration Service system.
- *password* is password you use to log in to the Integration Service system.

Uploading a Step with the API

The Integration Service system includes an API. When you install the Integration Service system, the API is available.

To upload a step with the API **`POST /steps`**:

```
POST /steps
{
  "name": "name_of_step",
  "data": "string"
}
```

where:

- *data* is all the information included in the step.

Validating Your Step

After uploading a step, you can use the API [POST /steps/run](#) to run the step individually without running an integration application. This allows you to validate that the step works as designed.

To run a step from the IS API:

```
POST /steps/run
{
  "name": "name_of_step",
  all other data from the .json file for the step
}
```

After the POST request is made, the Integration Service system will dispatch the step to a remote worker process for execution. By default, the POST request will wait five seconds for the step to complete. To override the default wait period, you can specify wait time as a parameter in the POST request. For example, to specify that the wait time should be 10 seconds :

```
POST /steps/run?wait=10
{
  "name": "example_step",
}
```

If the step completes within the wait time, the Integration Service system returns a 200 return code, logs, output, and the result of the step.

If the step does not complete within the wait time, the Integration Service system returns a task ID. You can use this task ID to view the logs for the step.

The API returns one of the following codes:

- 200. Step executed and completed within the timeout period.
- 202. Step executed but did not complete with timeout period or user did not specify wait. Returned data includes task to query for the status of the step
- 400. Required parameter for the step is missing.
- 404. Step not found.
- 500. Internal error. Database connection might be lost.

Viewing Logs

After running a step, you can view the log information for a step. Log information for a step is saved for the duration of the **result_expires** setting in the Integration Service system. The **result_expires** setting is defined in the file `opt/iservices/scripts/docker-compose.yml`. The default value for log expiration is 24 hours.

NOTE: To view the log information for a step before running an integration, you can use the API [POST /steps run](#) to run the step individually without running an integration application. You can then use the information in step #3 - step #6 below to view step logs.

To view the log information for a step:

1. Run an integration application.
2. Using an API tool like Postman or cURL, use the API [GET /applications/{appName}/logs](#):

```
GET URL_for_your_Integration_Service_system/api/v1/applications/integration_application_name/logs
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.

3. You should see something like this:

```
{
  "app_name": "example_integration",
  "app_vars": {},
  "href": "/api/v1/tasks/isapp-af7d3824-c147-4d44-b72a-72d9eae2ce9f",
  "id": "isapp-af7d3824-c147-4d44-b72a-72d9eae2ce9f",
  "start_time": 1527429570,
  "state": "SUCCESS",
  "steps": [
    {
      "href": "/api/v1/tasks/2df5e7d5-c680-4d9d-860c-e1ceccd1b189",
      "id": "2df5e7d5-c680-4d9d-860c-e1ceccd1b189",
      "name": "First EM7 Query",
      "state": "SUCCESS",
      "traceback": null
    },
    {
      "href": "/api/v1/tasks/49e1212b-b512-4fa7-b099-ea6b27acf128",
      "id": "49e1212b-b512-4fa7-b099-ea6b27acf128",
      "name": "second EM7 Query",
      "state": "SUCCESS",
      "traceback": null
    }
  ],
  "triggered_by": [
    {
      "application_id": "isapp-af7d3824-c147-4d44-b72a-72d9eae2ce9f",
      "triggered_by": "USER"
    }
  ]
}
```

4. In the "steps" section, Notice the lines that start with **href** and **id**. You can use these lines to view the logs for the application and the steps.

5. To use the **href** value to get details about a step, use an API tool like Postman or cURL and then use the API [GET /steps{step_name}](#):

```
GET URL_for_your_Integration_Service_system/href_value
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.
- *href_value* is the href value you can copy from the log file for the integration application. The href value is another version of the step name.

NOTE: To view logs for subsequent runs of the integration application, you can include the href specified in the **last_run** field.

6. To use the task **id** value to view details about a step, use an API tool like Postman or cURL and then use the API [GET /tasks/{task_ID}](#):

```
GET URL_for_your_Integration_Service_system/task_id
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.
- *task_id* is the id value you can copy from the log file for the integration application. The task ID specifies the latest execution of the step.

NOTE: After you find the href and task ID for a step, you can use those values to retrieve the most recent logs and status of the step.

Chapter

3

Creating an Integration Application

Overview

This chapter explains how to create your own integration application that can run on an Integration Service system.

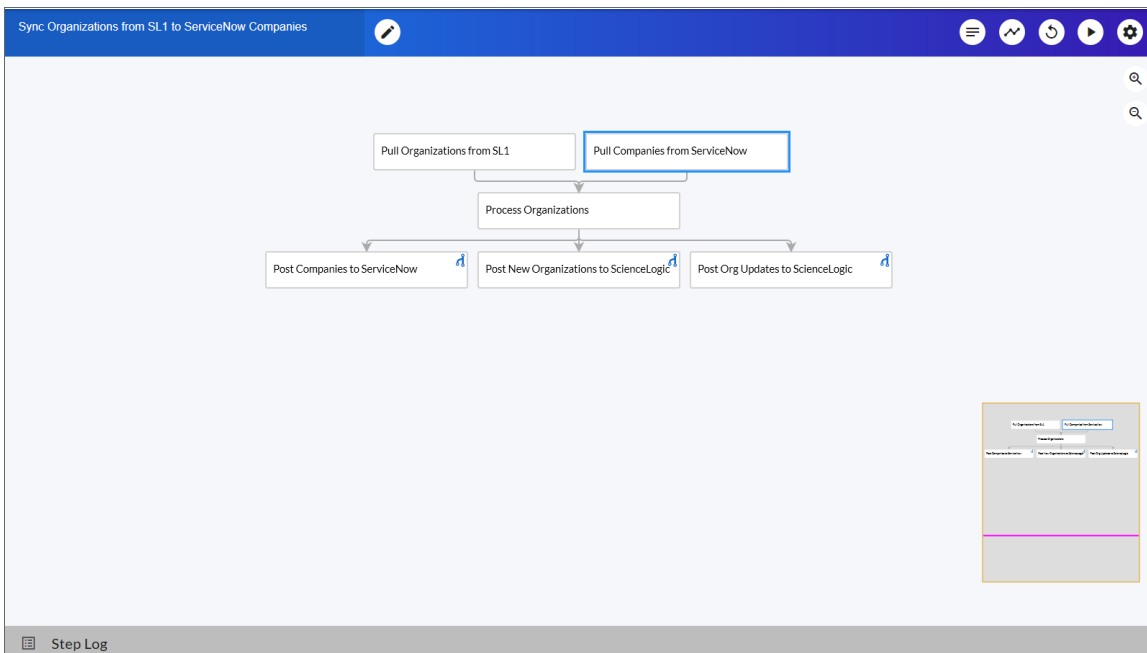
This chapter covers the following topics:

<i>What is an Integration Application?</i>	26
<i>Default Integration Application</i>	27
<i>Requirements</i>	27
<i>Creating an Integration Application from the Integration Template</i>	27
<i>Defining Required Fields for the Integration Application</i>	28
<i>Creating the List of Steps and Step Parameters</i>	29
<i>Transferring Data Between Steps</i>	30
<i>Defining Retry Options for a Step</i>	32
<i>Defining Variables for an Integration Application</i>	33
<i>Uploading the Integration Application to the Integration Service System</i>	34
<i>Running the Integration Application</i>	36
<i>Running the Integration Application with a Custom Queue</i>	36
<i>Defining a Configuration</i>	40
<i>Uploading the Configuration File to the Integration Service System</i>	42
<i>Viewing a Report for an Integration Application</i>	44
<i>Viewing Logs for an Integration Application</i>	46
<i>Working with Integration Applications in the Integration Service User Interface</i>	47

What is an Integration Application?

In the Integration Service, an **integration application** is a JSON file that specifies which steps to execute and the order in which to execute those steps. An integration application also defines variables and provides arguments for each step.

The following is an example of an integration application:



Integration application JSON objects are defined by configuration settings, steps that make up the integration, and application-wide variables used as parameters for each step. The parameters of each step can be configured dynamically, and each step can be named uniquely while still sharing the same underlying class, allowing for maximum re-use of code.

Integration applications can be executed through the REST API and are processed as an asynchronous task in the Integration Service. During processing the user is provided a unique task ID for the application and each of its tasks. Using the task IDs, the user can poll for the status of the integration application and the status of each individual running step in the integration application.

Executing an integration application from the REST API allows the user to dynamically set one-time parameter values for the variables defined in the integration.

The required parameters of integration applications are strictly enforced, and the Integration Service will refuse to execute the integration application if all required variables are not provided.

Default Integration Application

The Integration Service system includes a pre-defined integration application called *integration_template*. This default integration appears in the Integration Service user interface as the **Template App** integration application.

To view this integration application:

1. Using an API tool like Postman or cURL, use the API [GET /applications/{application_name}](#):

```
GET URL_for_your_Integration_Service_system/api/v1/applications/integration_template
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.

Requirements

To create an integration application, you must perform these tasks:

1. Download or copy the application template, called *integration_template*.
2. Define the required fields for the integration application.
3. Create the list of steps and step parameters.
4. Define application variables.
5. Upload the integration applet to the Integration Service.
6. Validate and test the integration applications step.

Creating an Integration Application from the Integration Template

The easiest way to create a new integration application is to use the application template that is included with the Integration Service system. To copy this template to your desktop:

1. Using an API tool like Postman or cURL, use the API [GET /applications/{application_name }](#):

```
GET URL_for_your_Integration_Service_system/api/v1/applications/integration_template
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.

2. Select and copy all the text in the application.
3. Open a source-code editor and paste the application in the source-code editor.
4. Save the new file as *newfilename.json*

where:

- *newfilename.json* is the new name of the integration application includes the .json suffix. The file name must be unique within your Integration Server system and cannot contains spaces.

Defining Required Fields for the Integration Application

Each integration application should include the following key:value pairs

```
"author": "name of the author",
"configuration": "if this integration application uses configuration variables,
specify the name of the configuration",
"description": "description of the integration application",
"friendly_name": "name that appears in the user interface",
"name": "file name without the py suffix".
"version": "version number",
```

The `integration_template` includes these required fields. You can edit the value for each key. To do so:

1. Using an API tool like Postman or cURL, use the API [GET /applications/{application_name}](#):

```
GET URL_for_your_Integration_Service_system/api/v1/applications/application_name
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.
- *application_name* is the integration application you want to edit.

2. Copy the contents of the integration application to a source-code editor.
3. Search for the section of the file that contains the text "description".
4. Supply new values for the following keys:

- **author**. User who created the integration template.
- **configuration** (if applicable). Configuration variables are defined in a stand-alone file (called a configuration) that lives on the Integration Service system and can be accessed by all integration applications and their steps. If your integration application or the steps in the application reference configuration variables, you must specify the name of the configuration in this value. For more details on creating a configuration, see the section on [Defining a Configuration](#).
- **description**. This description will be displayed in the user interface when viewing the available Integrations.
- **friendly_name**. The name of the integration application as it will appear in the user interface.

- **name**. Name of the file, without the .py suffix.
 - **Version**. Version number of the integration application.
5. Save your changes. Save the file to the same name (*application_name.json*.)
 6. To upload the integration application to the Integration Service system, see the section on [Uploading an Integration Application](#).

Creating the List of Steps and Step Parameters

In the integration application, you must specify the steps to execute and the order in which they should be executed.

If steps do not have dependencies, the Integration Service system will execute steps in parallel. If steps have dependencies (meaning one of the steps requires data from another step), the Integration Service system will execute the step that provides data and then execute the step that consumes that data.

To edit the steps section of the integration application:

1. Using an API tool like Postman or cURL, use the API [GET /applications/{application_name}](#):

```
GET URL_for_your_Integration_Service_system/api/v1/applications/application_name
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.
 - *application_name* is the integration application you want to edit.
2. Copy the contents of the integration application to a source-code editor.
 3. Search for the section of the file that contains the text "steps".
 4. The section of the integration application that specifies steps should look like this:

```
"steps": [  
  {  
    "name": "GETgoogle.com",  
    "file": "QueryREST",  
    "prefix_url": "http://google.com",  
    "method": "GET",  
    "output_to": ["next_step"]  
  },  
  {  
    "name": "next_step",  
    "file": "someOtherTask"  
  }  
]
```

5. Edit the "steps" section as needed and save your changes. Be sure to your file as a .JSON file with the same name as the integration application your downloaded.

6. To upload the integration application to the Integration Service system, see the section on [Uploading an Integration Application](#).

Specifying Values for name and file

For each step, you must specify its name and the file it executes. These two keys can have the same value.

- **name**. The name for the step. Other steps in the integration application can use this name to refer to the step. This value can include spaces. This name must be unique to the integration application.
- **file**. The name of the file that will be executed by the step. You could write step logic in a single file but use that step logic with different integration applications and use different names for the step in each integration application.

Specifying the Parameters

When you add a step to an integration application, you must view the step and determine if it includes any required parameters.

To view details about a step (in our example, the step file is named QueryREST):

1. Using an API tool like Postman or cURL, use the API `GET /steps/{step_name}`:

```
GET URL_for_your_Integration_Service_system/api/v1/steps/QueryREST
```

where:

- `URL_for_your_Integration_Service_system` is the IP address or URL for the Integration Service system.
2. The step GETgoogle.com (and its parent file, QueryREST) includes the following parameters that are defined as "required":

```
self.new_step_parameter(name=METHOD, sample_value="GET", description="HTTP method to perform", required=True)

self.new_step_parameter(name=PREFIX_URL, description="used with relative_url to create the full URL.", sample_value="http://10.2.11.253", default_value=None, required=True)
```

3. Therefore, when the integration application includes the step GETgoogle.com, the integration application must supply values for the required parameters.

```
"method": "GET",
"prefix_url": "http://google.com",
```

Transferring Data Between Steps

An essential part of integrations is passing data between tasks. The Integration Service system includes native support for saving and transferring Python objects between steps. Within a step, you can use one of the functions included in the `ipaa.score.BaseStep` class.

In an integration application, you can use the **output_to** key to specify that the results of a step should be piped to one or more specified steps. The `output_to` key uses the following syntax:

```
"output_to": ["step_name1", "step_name2"]
```

To pass data from step to step, you must include the `output_to` key along with the parameters for a step.

To edit the step parameters in the integration application:

1. Using an API tool like Postman or cURL, use the API [GET /applications/{application_name}](#):

```
GET URL_for_your_Integration_Service_system/api/v1/applications/application_name
```

where:

- `URL_for_your_Integration_Service_system` is the IP address or URL for the Integration Service system.
- `application_name` is the integration application you want to edit.

2. Copy the contents of the integration application to a source-code editor.
3. If you wanted the step "GETgoogle.com" to pass its output to the step "next-step", the integration application would include the following :

```
"steps": [  
  {  
    "name": "GETgoogle.com",  
    "file": "QueryREST",  
    "prefix_url": "http://google.com",  
    "method": "GET",  
    "output_to": ["next_step"]  
  },  
  {  
    "name": "next_step",  
    "file": "someOtherTask"  
  }  
]
```

- If steps have dependencies , the Integration Service system will first analyze all steps in an integration application and ensure that data-gathering steps are performed before steps that require that data.
4. Edit the "steps" section as needed and save your changes. Be sure to your file as a .JSON file with the same name as the integration application your downloaded.
 5. To upload the integration application to the Integration Service system, see the section on [Uploading an Integration Application](#).

Defining Retry Options for a Step

The following parameters allows you to define multiple retry options for a step. You can specify that the Integration Service system try to re-run a step if that step fails. Retries work following the rules of exponential backoff: the first retry will have a delay of 1 second, the second retry will have a delay of 2 seconds, the third retry will delay 4 seconds, the fourth retry will delay 8 seconds, and so on. You can edit the **retry_max** and **retry_backoff_max** values, below to further refine the retry settings.

You can include the following retry options in the integration application file, where you define parameters for each step:

- **retry_max** . The maximum number of times the Integration Service system will retry to execute the step before it stops retrying and logs a step failure. For example, if **retry_max** is 3, the Integration Service will retry after 1 second, then 2 seconds, then 4 seconds, and stop if the last retry fails. Possible values are:
 - Integer that specifies number of retries.
 - The default value is "0" (zero).
- **retry_backoff**. Instead of using a defined interval between retries, the Integration Service system will incrementally increase the interval between retries. Possible values are:
 - True
 - False
 - The default value is "False".
- **retry_jitter**. Instead of using a defined interval between retries, the Integration Service system will retry the step execution at random intervals. Possible values are:
 - True
 - False
 - The default value is "False".
- **retry_backoff_max**. The maximum time interval for the **retry_backoff** option, in seconds. For example, This means, if you have **retry_max** set to 15, the delays will be 1, 2, 4, 8, 16, 32, 64, 120, 240, 480, 600, 600, 600, and 600. Possible values are:
 - Integer that specifies number of seconds
 - The default value is "600"
- **retry_countdown**. The interval between retries, in seconds. If you enabled **retry_backup**, the Integration Service system will incrementally increase this interval:
 - Integer that specifies number of seconds
 - The default value is "180"

To define the retry options for a step in the integration application file:

1. Using an API tool like Postman or cURL, use the API GET `/applications/{application_name}`:

```
GET URL_for_your_Integration_Service_system/api/v1/applications/application_name
```

where:

- `URL_for_your_Integration_Service_system` is the IP address or URL for the Integration Service system.
- `application_name` is the integration application you want to edit.

2. Copy the contents of the integration application to a source-code editor.
3. If you wanted to add retry options to the step "GETgoogle.com", you could include the following:

```
"steps": [  
  {  
    "name": "GETgoogle.com",  
    "file": "QueryREST",  
    "prefix_url": "http://google.com",  
    "method": "GET",  
    "retry_max": 5  
    "retry_backoff": True  
    "retry_backoff_max": 600  
    "retry_countdown": 120  
    "output_to": ["next_step"]  
  },  
  {  
    "name": "next_step",  
    "file": "someOtherTask"  
  }  
]
```

4. Edit the "steps" section as needed and save your changes. Be sure to your file as a .JSON file with the same name as the integration application your downloaded.

Defining Variables for an Integration Application

Application variables are defined in the integration application. These variables can be accessed only by steps in the integration application. In a step, including the **appvar.** prefix in a variable tells the Integration Service system to look in the integration application to resolve the variable. In a step, application variables are used in step parameters.

To define an application variable:

1. Using an API tool like Postman or cURL, use the API `GET /applications/{application_name}`:

```
GET URL_for_your_Integration_Service_system/api/v1/applications/application_name
```

where:

- `URL_for_your_Integration_Service_system` is the IP address or URL for the Integration Service system.
- `application_name` is the integration application you want to edit.

2. Copy the contents of the integration application to a source-code editor.
3. Search for the section of the file that contains the text "app_variables".
4. In the integration_template file, the section looks like this:

```
"app_variables": [  
  {  
    "name": "exampleVariable", "value": "exampleValue", "description": "Variables  
    defined here will be available in all steps of an application", "required":  
    true, "sample_value": "SampleValue", "default_value": "DefaultValue"  
  }  
],
```

5. You can copy and paste the example section above for each application variable you want to define.
6. For each application variable, supply values for the following keys:
 - **name**. This key requires a value. The name of the variable.
 - **value**. This key requires a value. This is the value that the variable will resolve to at runtime.
 - **description**. A description of this variable. This description appears in the user interface. The default value is null.
 - **required**. Specifies whether the variable is required. Possible values are "True" or "False". The default value is "False". If the variable is required, but a value is not specified and a default value is not specified, the Integration Application will fail.
 - **sample_value**. A sample value of the required data type or schema. The default value is null. For example, if the parameter is "port" and the expected value is an integer, a good **sample_value** would be "443".
 - **default_value**. If no value is specified for this variable (in the **value** key), use the value of this key. The default value of this key is null.
7. Edit the "app_variables" section as needed and save your changes. Be sure to your file as a .JSON file with the same name as the integration application your downloaded.
8. To upload the integration application to the Integration Service system, see the section on [Uploading an Integration Application](#).

Uploading the Integration Application to the Integration Service System

When you create a new integration application or edit an existing integration application, you must upload the integration application to the Integration Service system. There are two ways to upload an integration application to the Integration Service system:

- at the command line with the iscli tool
- with the Integration Service API

Uploading an Integration Application with iscli

The Integration Service system includes a command-line utility called **iscli**. When you install Integration Service system, **iscli** is automatically installed.

To upload an integration application to the Integration Service system using iscli:

1. Either go to the console of the Integration Service system or use SSH to access the server.
2. Log in as **isadmin** with the appropriate password.
3. Enter the following at the command line:

```
iscli -u -a -f path_and_name_of_integration_application_file.py -H hostname_or_IP_address_of_integration_service_system -P port_number_of_http_on_integration_service_system -U user_name -p password
```

where:

- *name_of_integration_application_file* is the full pathname for the integration application.
- *hostname_or_IP_address_of_integration_service_system* is the hostname or IP address of the Integration Service system.
- *port_number_of_http_on_integration_service_system* is the port number to access the Integration Service system. The default value is 443.
- *user_name* is the user name you use to log in to the Integration Service system.
- *password* is password you use to log in to the Integration Service system.

Uploading an Integration Application with the API

The Integration Service system includes an API. When you install the Integration Service system, the API is available.

To upload an integration application with the IS API:

```
POST /applications
{
  "name": "name_of_integration_application",
  contents of the .json file for the integration application
}
```

The API returns one of the following:

- 200. Application successfully updated or added.
- 400. Name or data parameter is missing.
- 500. Internal error. Database connection might be lost.

Running the Integration Application

After uploading an integration application, you can run it to ensure that it works as designed. To run an integration application from the IS API:

1. Using an API tool like Postman or cURL, use the API `POST/applications/{application_name}/run`:

```
POST https://URL_for_your_Integration_Service_system/api/v1/applications/run
```

where:

- `URL_for_your_Integration_Service_system` is the IP address or URL for the Integration Service system.
- `application_name` is the integration application you want to edit.

2. The body of the post should contain:

```
{
  "name": "name_of_integration_application",
  "params": {}
}
```

3. You can include the following parameters with the POST request:

- parameters use the syntax:

```
"parameter name": "parameter value"
```

- **wait**. Number of seconds to wait for integration application to complete.
- **configuration**. The configuration file to use with the integration application. For details, see the section on [configurations](#).

4. The API returns one of the following:

- 200. Application started successfully.
- 400. Name or data parameter is missing.
- 404. Application not found.
- 500. Internal error. Database connection might be lost.

Running the Integration Application with a Custom Queue

If your Integration Service system runs multiple integration applications and you might want to ensure that one of those integration applications runs before other integration applications in the processing queues. To do this, you can align a queue with a worker process; the worker process will be dedicated to only that queue, will listen only to that queue, and process only jobs from that queue. You can then configure the high-priority integration application to use the queue that you aligned with its own worker process.

There are two steps to using a custom queue:

- Defining the queue and aligning it with a worker
- Configuring the Integration Application to use the custom queue

Defining a Custom Queue

To create a custom queue:

1. SSH to the Integration Service system
2. Use a text editor like vi to edit the file `/opt/iservices/scripts/docker-compose.yml`.
3. The `docker-compose.yml` file contains definitions for worker processes. For example, you might see something like this:

```
services:
  steprunner:
    image: sciencelogic/is-worker:latest
    environment:
      LOGLEVEL: 10
      celery_log_level: 10
      logdir: /var/log/iservices
      broker_url: 'pyamqp://guest@rabbit//'
      result_backend: 'redis://redis:6379/0'
      db_host: 'couchbase,localhost'
    secrets:
      - is_pass
      - encryption_key
    deploy:
      replicas: 2
    networks:
      - isnet
    depends_on:
      - redis
      - rabbitmq
      - couchbase
    volumes:
      - "/var/log/iservices:/var/log/iservices"
      - "statedb:/var/run/celery/states/"

  steprunner_1:
    image: sciencelogic/is-worker:latest
    environment:
      LOGLEVEL: 10
      celery_log_level: 10
      logdir: /var/log/iservices
      broker_url: 'pyamqp://guest@rabbit//'
      result_backend: 'redis://redis:6379/0'
      db_host: 'couchbase,localhost'
    user_queues: 'test_queue'
    secrets:
      - is_pass
      - encryption_key
    deploy:
      replicas: 2
    networks:
      - isnet
    depends_on:
      - redis
      - rabbitmq
```

```

- couchbase
volumes:
- "/var/log/iservices:/var/log/iservices"
- "statedb:/var/run/celery/states/"

steprunner_2:
  image: sciencelogic/is-worker:latest
  environment:
    LOGLEVEL: 10
    celery_log_level: 10
    logdir: /var/log/iservices
    broker_url: 'pyamqp://guest@rabbit//'
    result_backend: 'redis://redis:6379/0'
    db_host: 'couchbase,localhost'
  user_queues: 'critical_queue'
  secrets:
    - is_pass
    - encryption_key
  deploy:
    replicas: 2
  networks:
    - isnet
  depends_on:
    - redis
    - rabbitmq
    - couchbase
  volumes:
    - "/var/log/iservices:/var/log/iservices"
    - "statedb:/var/run/celery/states/"

```

4. The services with names that start with "steprunner" are the workers for the Integration Service system.

- Notice that the service named "steprunner" does not include any queues. This means that the worker "steprunner" listens to the default queues in the Integration Service system.

NOTE: ScienceLogic recommends that you allot at least one worker to handle the default queues.

- To add additional services to your Integration Service system. copy all the lines included in the service definition, paste it into the file, and rename the service.
- For example, you could copy all the lines from "steprunner_2", paste the text in the "services" section, and rename the service "steprunner_3".

5. To create one or more queues and dedicate a worker to them, enter the following line in definition of the worker, under the **environment** section:

```
user_queues: 'queue_name1, queue_name2'
```

where:

- *queue_name1*, *queue_name2*. The name of the new queues. The worker will monitor only these queues and execute tasks only from these queues.

6. After you have updated the docker-compose file, you can update and re-deploy the Integration Service system to pick up the changes to the docker-compose.yml file. To do this, SSH to the Integration Service system and execute the following command:

```
docker stack deploy -c /opt/iservices/scripts/docker-compose.yml is4
```

Configuring an Integration Application to Use a Custom Queue

There are two ways to specify that an integration application should use a custom queue:

- In the .JSON file for an integration application, you can specify that that integration application should always use a custom queue.
- At run time, you can specify that an integration application should use a custom queue only for that single execution of the integration application.

Configuring an Integration Application to Always Use a Custom Queue.

To specify that an integration application should always use a custom queue, edit the .json file for the integration application. To do this:

1. Using an API tool like Postman or cURL, use the API [GET /applications/{application_name}](#).

```
GET URL_for_your_Integration_Service_system/api/v1/applications/application_name
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.
- *application_name* is the integration application you want to edit.

2. Copy the contents of the integration application to a source-code editor.
3. Search for the section of the file that contains the lines "author", "configuration", "description", "friendly_name", and "name".
4. In that section of the file, added the bolded line:

```
"queue": "name_of_queue",
```

where:

- *name_of_queue* is one of the custom queues you defined in the docker-compose.yml file.

5. For example:

```
"author": "ScienceLogic, Inc.",  
"configuration": "",  
"description": "Read SL1 and ServiceNow devices and write them to a cache.",  
"friendly_name": "Cache SL1 Devices using GraphQL",  
"name": "cache_em7_device_data",  
"generate_report": true,  
"queue": "test_queue",
```

6. Save your changes. Save the file to the same name (*application_name.json*.)
7. The integration application will now always use the specified queue.
8. To upload your changes to the Integration Service system, see the section on [Uploading an Integration Application](#).

Configuring an Integration Application to Use a Custom Queue at Run Time

After uploading an integration application, you can run it and specify a queue. To run an integration application from the IS API:

1. Using an API tool like Postman or cURL, use the API [POST/applications/{application_name}/run](#):

```
POST https://URL_for_your_Integration_Service_system/api/v1/applications/run
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.
- *application_name* is the integration application you want to edit.

2. The body of the post should contain:

```
{
  "name": "name_of_integration application",
  "params": {
    "queue": "custom_queue"
  }
}
```

3. You can include the following parameter with the POST request:

```
"queue": "name_of_queue"
```

where:

- *name_of_queue* is one of the custom queues you defined in the `docker-compose.yml` file.

4. The API returns one of the following:

- 200. Application started successfully.
- 400. Name or data parameter is missing.
- 404. Application not found.
- 500. Internal error. Database connection might be lost.

Defining a Configuration

Configuration variables are defined in a stand-alone JSON file called a configuration that lives on the Integration Service system and can be accessed by all integration applications and their steps.

In a step, including the **config.** prefix with a variable tells the Integration Service system to look in a configuration file to resolve the variable.

If you want to re-use the same settings (like hostname and credentials) between applications, define configuration variables.

To define a configuration file:

1. Use a source-code editor to open a new file.
2. Copy and paste the following example text into the new file:

```
{
  "author": "ScienceLogic, Inc.",
  "description": "Fsun's config for demo.",
  "name": "fsun-demo-settings",
  "friendly_name": "Demo Settings",
  "configuration_data": [
    {
      "encrypted": false,
      "name": "em7_host",
      "value": "10.2.11.42"
    },
    {
      "encrypted": false,
      "name": "em7_user",
      "value": "em7admin"
    },
    {
      "encrypted": true,
      "name": "em7_password",
      "value": "+dqGJe1NwTyvda02EizTWjJ2uj2C1wzBzgNqVhpdTHA="
    },
  ],
}
```

3. To create your own configuration file, edit the following keys:
 - **author**. Name of the author of the configuration file. This field is optional.
 - **description**. Description of the configuration file. This field is optional.
 - **name**. Name of the configuration file. The name cannot contain any spaces and should be unique in your Integration Service system
 - **friendly_name**. User-friendly name for the configuration.
4. The section under "configuration_data" defines each configuration variable. You can use these variables as examples.
5. To create your own configuration variables, note the syntax:
 - The variable definition is surrounded by curly braces
 - Each key:value pair in the definition is separated with a comma.

6. To create your own configuration variable, define the following keys:
 - **encrypted**. Specifies whether the value will appear in plain text or encrypted in this .json file. If set to "true", specifies that when the value is uploaded, the Integration Service system will encrypt value of the variable. The plain text value will never again be retrievable by an end user. The encryption key is unique to each Integration Service system.
 - **name**. Name of the configuration file (without the .json suffix). This value appears in the user interface.
 - **value**. The value to assign to the variable.
7. Repeat steps 5 and 6 for each configuration variable.
8. Save the new file as *config_file.json*
where:
 - *config_file.json* is the name of the configuration file. This value must match the **name** key in the file. The file name must be unique within your Integration Server system and cannot contains spaces.

Uploading the Configuration File to the Integration Service System

When you create a new configuration file or edit an existing configuration file, you must upload the step to the Integration Service system. There are two ways to upload a configuration file to the Integration Service system:

- at the command line with the *iscli* tool
- with the Integration Service API

Uploading a Configuration with *iscli*

The Integration Service system includes a command line tool called *iscli*. When you install Integration Service system, *iscli* is automatically installed.

To upload a configuration file to the Integration Service system using *iscli*:

1. Either go to the console of the Integration Service system or use SSH to access the server.
2. Log in as *isadmin* with the appropriate password.

3. Type the following at the command line:

```
iscli -u -c -f path_and_name_of_configuration_file.py -H hostname_or_IP_address_of_integration_service_system -P port_number_of_http_on_integration_service_system -U user_name -p password
```

where:

- *path_and_name_of_configuration_file* is the full pathname for the step.
- *hostname_or_IP_address_of_integration_service_system* is the hostname or IP address of the Integration Service system.
- *port_number_of_http_on_integration_service_system* is the port number to access the Integration Service system. The default value is 443.
- *user_name* is the user name you use to log in to the Integration Service system.
- *password* is password you use to log in to the Integration Service system.

Uploading a Configuration with the API

The Integration Service system includes an [API](#). When you install the Integration Service system, the API is available.


To upload a configuration with the IS API:

```
POST /configurations
{
  contents of the .json file for the configuration
}
```

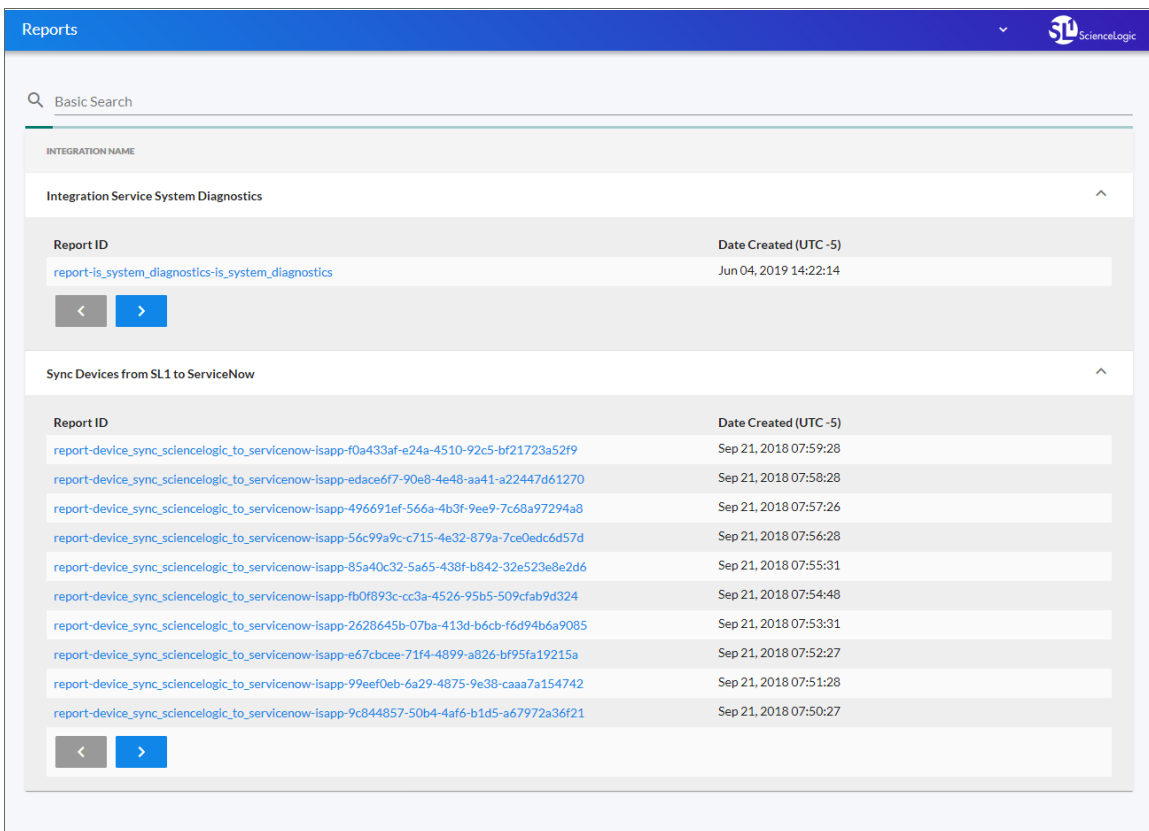
The API returns one of the following:

- 200. Configuration successfully updated or added.
- 400. Name or data parameter is missing.
- 500. Internal error. Database connection might be lost.

Viewing a Report for an Integration Application

In the Integration Service user interface, the **Reports** page () contains a list of reports associated with integration applications. If an integration application has the reporting feature enabled and the application supports reports, then the Integration Service will generate a report each time you run the integration application.

An individual report displays data only from the most recent run of the integration application; a report is not an aggregation of all previous runs.



The screenshot shows the 'Reports' page in the Integration Service user interface. The page has a blue header with the 'ScienceLogic' logo. Below the header is a search bar labeled 'Basic Search'. The main content area is divided into two sections, each with a title and a list of reports.

Integration Service System Diagnostics

Report ID	Date Created (UTC -5)
report-is_system_diagnostics-is_system_diagnostics	Jun 04, 2019 14:22:14


Sync Devices from SL1 to ServiceNow

Report ID	Date Created (UTC -5)
report-device_sync_sciencelogic_to_servicenow-isapp-f0a433af-e24a-4510-92c5-bf21723a52f9	Sep 21, 2018 07:59:28
report-device_sync_sciencelogic_to_servicenow-isapp-edace6f7-90e8-4e48-aa41-a22447d61270	Sep 21, 2018 07:58:28
report-device_sync_sciencelogic_to_servicenow-isapp-496691ef-566a-4b3f-9ee9-7c68a97294a8	Sep 21, 2018 07:57:26
report-device_sync_sciencelogic_to_servicenow-isapp-56c99a9c-c715-4e32-879a-7ce0edc6d57d	Sep 21, 2018 07:56:28
report-device_sync_sciencelogic_to_servicenow-isapp-85a40c32-5a65-438f-b842-32e523e8e2d6	Sep 21, 2018 07:55:31
report-device_sync_sciencelogic_to_servicenow-isapp-fb0f893c-cc3a-4526-95b5-509cfab9d324	Sep 21, 2018 07:54:48
report-device_sync_sciencelogic_to_servicenow-isapp-2628645b-07ba-413d-b6cb-f6d94b6a9085	Sep 21, 2018 07:53:31
report-device_sync_sciencelogic_to_servicenow-isapp-e67cbcee-71f4-4899-a826-bf95fa19215a	Sep 21, 2018 07:52:27
report-device_sync_sciencelogic_to_servicenow-isapp-99eef0eb-6a29-4875-9e38-caaa7a154742	Sep 21, 2018 07:51:28
report-device_sync_sciencelogic_to_servicenow-isapp-9c844857-50b4-4af6-b1d5-a67972a36f21	Sep 21, 2018 07:50:27

You can search for a specific report by typing the name of that report in the **Search** field at the top of the **Reports** page. The user interface filters the list as you type.

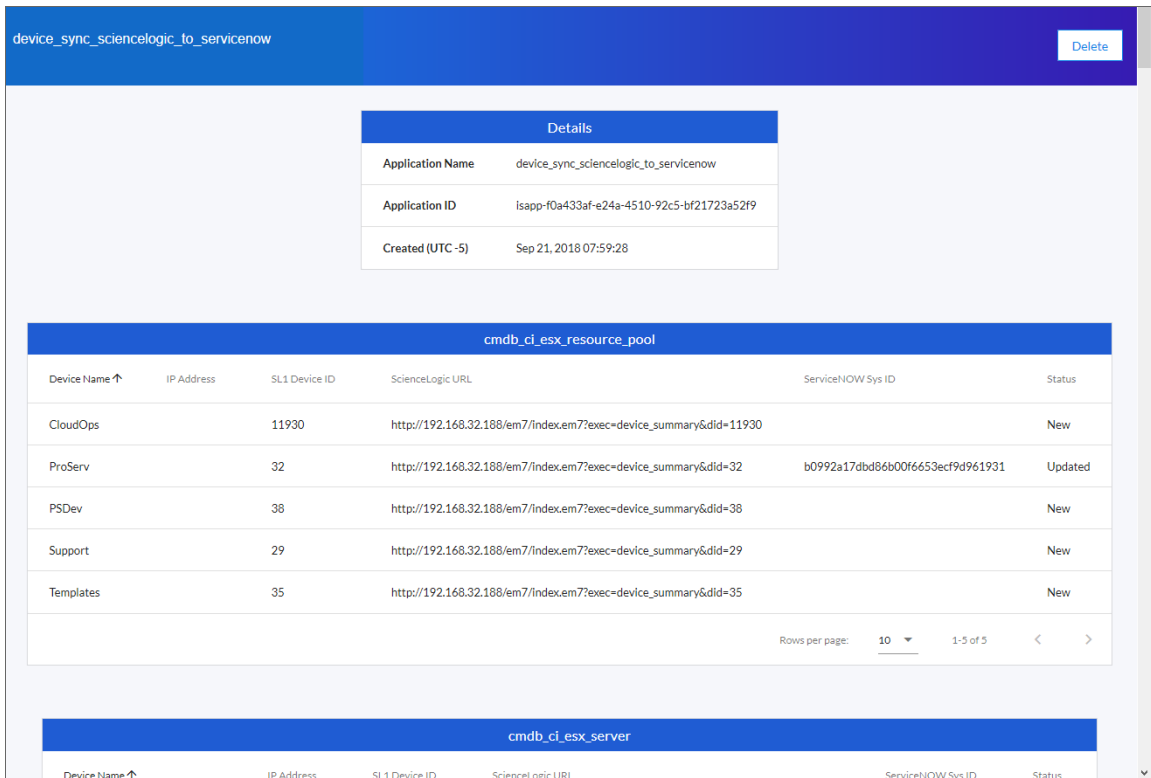
NOTE: Not all integration applications generate reports. Currently, only the "Integration Service System Diagnostics" integration applications support the generation of reports.

To view details for an integration application report:

1. On the **Reports** page (), click the name of the integration application to expand the list of reports for that application.

TIP: Click the arrow buttons to scroll forward and back through the list of reports.

2. Click a report name in the **Report ID** column. The **Report Details** page appears:



The screenshot shows the 'Report Details' page for the integration application 'device_sync_sciencelogic_to_servicenow'. At the top right, there is a 'Delete' button. Below the application name, a 'Details' table provides the following information:

Details	
Application Name	device_sync_sciencelogic_to_servicenow
Application ID	isapp-f0a433af-e24a-4510-92c5-bf21723a52f9
Created (UTC-5)	Sep 21, 2018 07:59:28

Below the details table, there is a table titled 'cmdb_ci_esx_resource_pool' with the following columns: Device Name, IP Address, SL1 Device ID, ScienceLogic URL, ServiceNOW Sys ID, and Status. The table contains five rows of data:

Device Name	IP Address	SL1 Device ID	ScienceLogic URL	ServiceNOW Sys ID	Status
CloudOps		11930	http://192.168.32.188/em7/index.em7?exec=device_summary&did=11930		New
ProServ		32	http://192.168.32.188/em7/index.em7?exec=device_summary&did=32	b0992a17dbd86b00f6653ecf9d961931	Updated
PSDev		38	http://192.168.32.188/em7/index.em7?exec=device_summary&did=38		New
Support		29	http://192.168.32.188/em7/index.em7?exec=device_summary&did=29		New
Templates		35	http://192.168.32.188/em7/index.em7?exec=device_summary&did=35		New

At the bottom of the table, there is a pagination control showing 'Rows per page: 10' and '1-5 of 5' with navigation arrows. Below this table, there is another table titled 'cmdb_ci_esx_server' with the same columns as the 'cmdb_ci_esx_resource_pool' table.

TIP: The **Status** field in a report displays the current state of the synced item, which can include *New, Removed, Updated, or Unchanged*.

3. To view the detail page for the integration application on the **Integrations** page, click the **Application Name** link.

TIP: From the detail page for the integration application, click **[Reports]** () to return to the **Reports** page.

4. To delete a report, open the report and click **[Delete]**. Click **[OK]** to delete the report.

Viewing Logs for an Integration Application

After running an integration application, you can view its log information. To view the log information for an integration application:

1. Run an integration application.
2. Using an API tool like Postman or cURL, use the API `GET /applications/{appName}/logs`:

```
GET URL_for_your_Integration_Service_system/api/v1/applications/integration_application_name/logs
```

where:

- `URL_for_your_Integration_Service_system` is the IP address or URL for the Integration Service system.

3. You should see something like this:

```
{
  "app_name": "example_integration",
  "app_vars": {},
  "href": "/api/v1/tasks/isapp-af7d3824-c147-4d44-b72a-72d9eae2ce9f",
  "id": "isapp-af7d3824-c147-4d44-b72a-72d9eae2ce9f",
  "start_time": 1527429570,
  "state": "SUCCESS",
  "steps": [
    {
      "href": "/api/v1/tasks/2df5e7d5-c680-4d9d-860c-e1ceccd1b189",
      "id": "2df5e7d5-c680-4d9d-860c-e1ceccd1b189",
      "name": "First EM7 Query",
      "state": "SUCCESS",
      "traceback": null
    },
    {
      "href": "/api/v1/tasks/49e1212b-b512-4fa7-b099-ea6b27acf128",
      "id": "49e1212b-b512-4fa7-b099-ea6b27acf128",
      "name": "second EM7 Query",
      "state": "SUCCESS",
      "traceback": null
    }
  ],
  "triggered_by": [
    {
      "application_id": "isapp-af7d3824-c147-4d44-b72a-72d9eae2ce9f",
      "triggered_by": "USER"
    }
  ]
}
```

4. Notice the bolded lines that start with `href` and `id`. You can use these lines to view the logs for the application.
5. To use the `href` value to get details about an application, use an API tool like Postman or cURL and use the API `GET /applications/{appName}`:

```
GET URL_for_your_Integration_Service_system/href_value
```


where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.
- *href_value* is the href value you can copy from the log file. The href value is another version of the application name.

NOTE: To view logs for subsequent runs of the integration application, you can include the href specified in the *last_run* field.

6. To use the task **id** value to view details about an application, use an API tool like Postman or cURL and use the API [GET /tasks/{task_ID}](#):

```
GET URL_for_your_Integration_Service_system/task_id
```

where:

- *URL_for_your_Integration_Service_system* is the IP address or URL for the Integration Service system.
- *task_id* is the id value you can copy from the log file for the integration application. The task ID specifies the latest execution of the application.

Working with Integration Applications in the Integration Service User Interface

You can log into the user interface for the Integration Service to edit, configure, and run an integration application.

Editing an Integration Application in the User Interface

On the **Integrations** page, you can edit an existing integration application and its steps. You can also add and remove steps from that application.


NOTE: You cannot overwrite integration applications where *ScienceLogic, Inc.* is listed as the Author, but you can edit a ScienceLogic application and save it with a different name.



To edit an integration application:

1. From the **Integrations** page, select the integration application that you want to edit. The **Integration Application** page for that application appears.
2. Click the **[Open Editor]** button (🔧). The **Steps Registry** pane appears with a list of all of the steps that are available for that integration application:

MODULE	DATE/TIME (UTC -5)	LOG LEVEL	MESSAGE
1 ipaas_logger	Nov 13, 2019 15:58:11, 574	FLOW	Start Collect IS system info
2 QueryREST	Nov 13, 2019 15:58:11, 682	ERROR	HTTP(s) Response Actual: 404 Expected: 200, does NOT match for: https://contentapi.isnet:5000/api/v1/applications/device_sync_sciencelogic_to_servicenow
3 QueryREST	Nov 13, 2019 15:58:16, 726	ERROR	HTTP(s) Response Actual: 404 Expected: 200, does NOT match for: https://contentapi.isnet:5000/api/v1/applications/device_sync_sciencelogic_to_servicenow
4 QueryREST	Nov 13, 2019 15:58:21, 771	ERROR	HTTP(s) Response Actual: 404 Expected: 200, does NOT match for: https://contentapi.isnet:5000/api/v1/applications/device_sync_sciencelogic_to_servicenow
5 QueryREST	Nov 13, 2019 15:58:26,	ERROR	HTTP(s) Response Actual: 404 Expected: 200, does NOT match for: https://contentapi.isnet:5000/api/v1/applications

3. Scroll through the **Steps Registry** pane or use the **Search** field at the top of the pane to find the step you want to add.

TIP: Click the **[Actions]** button () on a step to view more information about that step, including the step ID, the Synchronization PowerPack for that step, and the step version author. You can also click **[Edit Step Code]** to edit the code for that step.


4. Click the steps you want to add from the **Steps Registry** pane and drag them to the main viewing pane of the **Integration Application** page to add them to the integration application.
5. To adjust the position of any step in the integration application, click the step you want to move and drag it to its new location.
6. To edit the configuration for a step, click the gear icon () on the step and update the fields as needed. You can add test data to the step and click **Custom Run** to run test data for that step.
7. To redirect the arrows connecting the steps, click an arrow and drag it to reposition it.
8. To remove a step, click the step to select it and press the **[Delete]** key on your keyboard.
9. To save the changes you made to the integration application, click **[Save]**. You can also click the Save as option to save the application with a new name.
10. To stop editing and close the **Search Steps Registry** panel, click the **[Close Editor]** button ().


Configuring an Integration Application in the User Interface

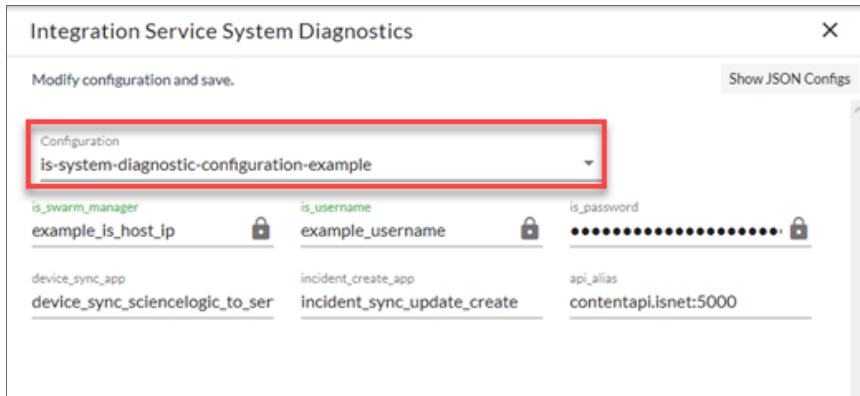
Before you can run an integration application, you must align the application with a configuration object from the **Configurations** page. A **configuration object** defines global variables, such as endpoints and credentials, that can be used by multiple steps and integration applications. Each variable in a configuration object is set up as a name and value pair. You can also encrypt a variable to protect sensitive data like a password.

You can "align" the configuration object you want to use with an integration application from the **Configuration** pane for that application integration.

To align a configuration object with an integration application:

1. From the **Integrations** page (), select the integration application that you want to align with a configuration object. The **Integration Application** page for that integration application appears.

2. Click **[Configure]** (). The **Configuration** pane opens on the right side of the **Integration Application** page:




TIP: To view a pop-up description of a field on the **Configuration** pane for an integration application, hover over the label name for that field.

3. Select a configuration from the **Configuration** dropdown to "align" to this integration application. This step is required for all integration applications.

TIP: You can select *none* from the **Configuration** dropdown to clear or "un-align" the selected configuration object from an integration application. Also, if you did not select a configuration object when editing fields on the **Configuration** pane, the previously set configuration object will remain aligned (if there was a previously set configuration object).

4. Click **[Show JSON Configs]** to view the JSON configuration data for the configuration object. Click **[Hide JSON Editor]** again to view the fields instead.
5. As needed, edit the other configuration values for the application. Press **[Enter]** after editing an item to make sure your changes are saved.

NOTE: To prevent potential issues with security and configuration, any fields that are encrypted and any configuration-specific fields containing a padlock icon () on the **Configuration** pane cannot be edited.



6. When you are finished, click **[Save]** at the bottom of the **Configuration** pane.


Running or Stopping an Integration Application in the User Interface

You can run an integration application directly from the **Integrations** page (the list view) or from the **Integration Application** page (the detail view). If you run the integration application from the **Integration Application** page, you have the following additional options:

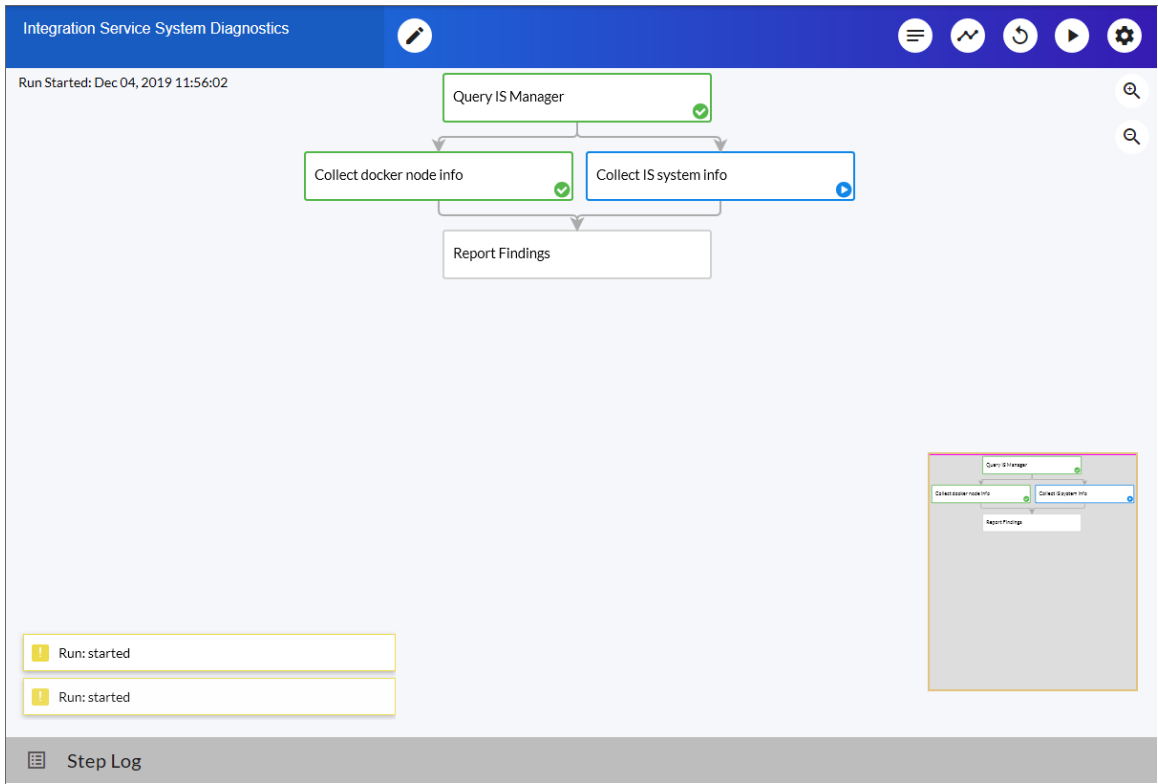
- **Run**. Executes the integration application normally, with a log level of 1. This is the default, and it is the same as the *Run Now* option from the **Integrations** page.
- **Debug Run**. Executes the integration application in Debug Mode with a log level of 10.
- **Custom Run**. Executes the integration application using a logging level that you specify (Error, Warning, Info, or Debug). You can also add any customer parameters that you might want to use to test specific features in the integration application.

To run an integration application:

1. From the **Integrations** page (), click the **[Actions]** button () for the integration application you want to run and select *Run Now*.

TIP: You can also select an application from the **Integrations** page and click **[Run]** () from the **Integration Application** page. If you hover over the **[Run]** button, you can select *Debug Run* or *Custom Run*.

2. As the application runs, the color of the border around each step represents whether it is running, is successful, or has failed:



Step Color	Icon	State
Blue		Running
Green		Successful
Red		Failed
Yellow		Warning

NOTE: Pop-up status messages also appear in the bottom left-hand corner of the **Integration Application** page to update you on the progress of the application run.

3. If a step triggers a child application, a branch icon (

Chapter

4

Creating a Synchronization PowerPack

Overview

This chapter describes how to package your integration applications, steps, configuration objects, and utilities into a Synchronization PowerPack.

NOTE: The label "SyncPack" is used in place of "Synchronization PowerPack" in the Integration Service user interface.

This chapter covers the following topics:

<i>What is a Synchronization PowerPack?</i>	54
<i>Generating the Synchronization PowerPack Structure</i>	54
<i>Building a Synchronization PowerPack</i>	56
<i>Uploading a Synchronization PowerPack to the Integration Service</i>	57
<i>Activating and Installing a Synchronization PowerPack</i>	58
<i>Extending an Existing Synchronization PowerPack</i>	59

What is a Synchronization PowerPack?

A **Synchronization PowerPack** is a Python `.whl` file that contains integration applications, steps, configuration objects, and utilities that can be used in the Integration Service. Synchronization PowerPacks are versioned and can define dependencies on other Synchronization PowerPacks or on external Python packages.

When applications from a Synchronization PowerPack are executed, they run within a Python Virtual Environment that contains the Synchronization PowerPack itself and its dependencies.

Generating the Synchronization PowerPack Structure

ScienceLogic uses a **Cookiecutter** repository for generating the standard Synchronization PowerPack structure. This structure includes all necessary files and the default directory structure that is generated based on your response to the prompts.

For more information about Cookiecutter, see <https://cookiecutter.readthedocs.io/en/latest/>.

To use Cookiecutter to generate the Synchronization PowerPack structure:

1. On a system with Python 3.6 or later installed, run the following command:

```
pip install cookiecutter
cookiecutter https://github.com/ScienceLogic/is_syncpack_cookiecutter.git
```

2. Complete the following fields:

- **author**. Name of the person or organization that created this Synchronization PowerPack.
- **url_project**. URL of the repository where this Synchronization PowerPack is hosted.
- **syncpack_name**. Developer name of the Synchronization PowerPack, without spaces. For example: `test_syncpack`.
- **syncpack_friendly_name**. Friendly name of the Synchronization PowerPack, which will display in the Integration Service user interface. For example: `Test Synchronization PowerPack`.
- **syncpack_description**. Short description of the Synchronization PowerPack, which will display in the Integration Service user interface and in the **readme** file.
- **version**. Number of the Synchronization PowerPack version.
- **requires_minimum_is_version**. The minimum version of the Integration Service platform required for this Synchronization PowerPack.

After you complete this process, Cookiecutter creates a new directory in your current working directory named with the **syncpack_name** you specified.

Synchronization PowerPack Structure

After you use Cookiecutter to generate the Synchronization PowerPack structure, you can access the following files and folders in the folder created by the previous process.

- **MANIFEST.in**. Allows the Python **.whl** file to contain non-Python files.
- **README.md**. The **readme** file for this Synchronization PowerPack, which is pre-populated with basic information.
- **docker**. This directory contains the following Docker information:
 - **Dockerfile**. Sample Dockerfile that can be used for building and testing the Synchronization PowerPack in a CI/CD environment.
 - **docker-compose.yml**. Docker Compose file that uses above container.
- **setup.py**. Python setup file for building the Synchronization PowerPack. You should not need to modify this file.
- **test_syncpack**. This directory contains all Synchronization PowerPack content, including the following:
 - **__init__.py**. Python init file.
 - **apps**. This directory contains Synchronization PowerPack integration application json files.
 - **configs**. This directory contains Synchronization PowerPack configuration object json files.
 - **[meta.json]**. This file defines the [Synchronization PowerPack properties](#).
 - **steps**. This directory contains Synchronization PowerPack Step Python files.
 - **util**. This directory contains Python classes or modules that can be used by multiple steps in the Synchronization PowerPack .
- **tox.ini**. Basic tox configuration for running unit tests.

Synchronization PowerPack Properties

The **meta.json** file contains all configurable properties for a Synchronization PowerPack. This file is where you define the name, friendly name, version, description, default schedules, tags, and supported Integration Service version of the Synchronization PowerPack.

`requires_dist` is where you define the Python packages on which this Synchronization PowerPack depends. For example, if you would like to use a step from the *Base Steps* Synchronization PowerPack and also use Amazon's boto library, you would use the following:

```
"requires_dist": ["boto", "base_steps_syncpack"],
```

The following code block contains an example of the **meta.json** file:

```
{  
  "name": "test_syncpack",  
  "version": "0.0.1",  
  "summary": "Testing Syncpack",  
  "description": "IS Syncpack for testing",  
}
```

```

"author": "John Smith",
"home_page": "https://example.com/test_syncpack",
"requires_dist": [],
"schedules": [
  {
    "application_id": "dummy_app",
    "entry_id": "dummy app",
    "params": {},
    "schedule": {
      "schedule_info": {
        "run_every": 3600
      },
      "schedule_type": "frequency"
    }
  }
],
"tags": ["test_syncpack"],
"critical": [],
"requires_is_version": ">=2.0.0"
}

```

Building a Synchronization PowerPack

After you complete your Synchronization PowerPack structure and place your integration applications, steps, and configuration objects in the right directories, you can build the Synchronization PowerPack **.whl** file (wheel).

You can build a Synchronization PowerPack with Python3 or with the `iscli` utility that comes with the Integration Service.

NOTE: Not included is how to build a Synchronization PowerPack automatically with CI/CD platforms.

Building a Synchronization PowerPack with Python3

To build the Synchronization PowerPack wheel with Python3, run the following command from the top-level Synchronization PowerPack directory:

```
python3 setup.py sdist bdist_wheel
```

This command creates **build** and **dist** folders within your SyncPack directory. In the **dist** folder, you will now have a Python wheel that you can upload to the Integration Service.

Building a Synchronization PowerPack with the `iscli` Utility

If you build your Synchronization PowerPack with the `iscli` utility, the utility builds and uploads the Synchronization PowerPack to the Integration Service in one step.

To build and upload your Synchronization PowerPack with the `iscli` utility, run the following command:

```
iscli -H <IS-IP-host> -U <IS User> -p <IS Password> -ukf </path/to/SyncPack/Directory>
```

where `</path/to/SyncPack/Directory>` is the Synchronization PowerPack root that contains the **setup.py** file.

Uploading a Synchronization PowerPack to the Integration Service

After you create your Synchronization PowerPack wheel, you can upload it to an Integration Service system. You can upload this Synchronization PowerPack to Integration Service using the Integration Service user interface or with the `iscli` utility.

Uploading a Synchronization PowerPack with the User Interface

NOTE: You must import and install the *ServiceNow Base Synchronization PowerPack* before uploading and installing any of the other ServiceNow Synchronization PowerPacks.

To import a Synchronization PowerPack in the Integration Service user interface:

1. On the **SyncPacks** page, click **[Import SyncPack]**. The **Import SyncPack** page appears.
2. Click **[Browse]** and select the `.whl` file for the Synchronization PowerPack you want to install.

TIP: You can also drag and drop a `.whl` file to the **SyncPacks** page.

3. Click **[Import]**. The Integration Service registers and uploads the Synchronization PowerPack. The Synchronization PowerPack is added to the **SyncPacks** page.

NOTE: You cannot edit the content package in a Synchronization PowerPack published by ScienceLogic. You must make a copy of a ScienceLogic Synchronization PowerPack and save your changes to the new Synchronization PowerPack to prevent overwriting any information in the original Synchronization PowerPack when upgrading.

Uploading a Synchronization PowerPack with the `iscli` Utility

To upload a Synchronization PowerPack with the `iscli` utility:

1. Using a tool like secure copy (SCP), copy the Synchronization PowerPack from your local computer to your Integration Service host.
2. Start an SSH session with your Integration Service host.
3. Run the following command:

```
iscli -ukf /path/to/test_syncpack-0.0.1-py2.py3-none-any.whl
```

NOTE: If you have already uploaded this Synchronization PowerPack to your Integration Service system and you want to re-upload with the same version, add the `-F` flag to the above command.


4. You can now activate the Synchronization PowerPack in the Integration Service. See the following topic for more information.


Activating and Installing a Synchronization PowerPack





After you upload a Synchronization PowerPack to the Integration Service, you need to activate and install the Synchronization PowerPack to access its integration applications, steps, and configuration objects.

Activating and Installing with the User Interface

To install a Synchronization PowerPack in the Integration Service user interface:

1. On the **SyncPacks** page, click the **[Actions]** button () for the Synchronization PowerPack you want to install and select *Activate & Install*. The **Activate & Install SyncPack** modal appears.

TIP: By default, the **SyncPacks** page displays only activated and installed PowerPacks. If you do not see the PowerPack that you want to install, click the toggle icon () on the **SyncPacks** page and select *Show All SyncPacks* to see a list of the uninstalled PowerPacks.

2. Click **[Yes]** to confirm the activation and installation. When the Synchronization PowerPack is activated, the **SyncPacks** page displays a green check mark icon () for that Synchronization PowerPack. If the activation or installation failed, then a red exclamation mark icon () appears.
3. For more information about the activation and installation process, click the check mark icon () or the exclamation mark icon () in the **Activated** column for that Synchronization PowerPack. For a successful installation, the "Activate & Install SyncPack" integration application appears, and you can view the Step Log for the steps. For a failed installation, the **Error Logs** window appears.

Activating and Installing with the iscli Utility

To activate and install an uploaded Synchronization PowerPack using the `iscli` utility, run the following command:

```
iscli -AIkn <syncpack_name> -V <syncpack_version>
```

where `<syncpack_name>` is the name from the `meta.json` file, and `<syncpack_version>` is the version number of the Synchronization PowerPack.

For example:

```
iscli -AIkn test_syncpack -V 0.0.1
```

NOTE: To force a re-activation of the Synchronization PowerPack, add `-F` to the above command.

Extending an Existing Synchronization PowerPack

Because the content that is included in a Synchronization PowerPack cannot be overwritten inside the Integration Service, you will need to extend existing Synchronization PowerPacks with your own Synchronization PowerPack to add customizations.

Restrictions

- The names for integration applications, steps, and configuration objects must be unique within the entire Integration Service platform. For example, you cannot have a step named "QueryREST", because that step is already included in the *Base Steps* Synchronization PowerPack. You will need to change the name of your customized step.
- Your new Synchronization PowerPack must include the Synchronization PowerPack you are extending in the **requires_dist** list in your **meta.json**. This ensures that your Synchronization PowerPack will have access to the integration applications and steps of the parent Synchronization PowerPack in its execution environment.
- After you install the Synchronization PowerPack, its environment will never be modified, even if the parent Synchronization PowerPacks are updated. If you wish to include changes from the parent Synchronization PowerPack, you will need to re-install your customized Synchronization PowerPack.

Modifying Integration Application Definitions

To change the API endpoint or query used in an existing integration application:

1. Copy the entire JSON file for the integration application from the original Synchronization PowerPack into the `apps` directory of your new Synchronization PowerPack.
2. Rename the file so it will not conflict with the existing integration application.
3. In the JSON code, change the `friendly_name`, `author`, `description`, and `version` as required.
4. Make any other changes to the integration application as desired.

Modifying a Step

You have two methods for modifying a step. The first method is to change the code in a step itself, and the second is to extend the step class and execute your own code before or after the existing step.

To change the logic performed within a step:

1. Copy the entire step Python file from the original Synchronization PowerPack into the `steps` directory of your new Synchronization PowerPack.
2. Rename the file so it will not conflict with the existing step.

3. In the step file, change the class name to match the name of the step file.
4. In the `__init__()` section of the code, change the `friendly_name`, `description`, and `version` as required.
5. Make any other changes to the step as desired.

To "extend" the behavior of an existing step by performing work before or after the existing logic:

1. Create a new step and give it a unique name.
2. Instead of importing and extending `BaseStep` import and extend the step class you would like to modify. For example:

```
from base_steps_syncpack.steps.QueryGQL import QueryGQL

class OptionalQueryGQL(QueryGQL):
```

3. Complete the `__init__()` as normal, but make sure to call `init` of the super class to include all of its parameters. For example:

```
super(OptionalQueryGQL, self).__init__()
```

4. Create your additional logic in the `execute()` function. This logic can be performed either before or after you call the `execute` of the parent step. For example:

```
super(OptionalQueryGQL, self).execute()
```

NOTE: If you would like to perform additional logic after the `execute` of the parent step runs, you can access the data from that step via `self.get_current_saved_data()`.

Full Extended Step Example

```
from base_steps_syncpack.steps.QueryGQL import QueryGQL
from ipaascore import parameter_types
from ipaascommon.ipaas_utils import str_to_bool

class OptionalQueryGQL(QueryGQL):

    def __init__(self):
        super(OptionalQueryGQL, self).__init__()
        self.friendly_name = "Optional QueryGQL Call"
        self.description = "Optionally Runs GQL call."
        self.version = "1.0.0"

        self.new_step_parameter(
            name="enable",
            description="Should this step run.",
            default_value=True,
            required=False,
            param_type=parameter_types.BoolParameterToggle()
        )

    def execute(self):
        enabled = str_to_bool(self.get_parameter("enable", None))
```

```
if enabled:
    super(OptionalQueryGQL, self).execute()

else:
    self.logger.flow(
        "Step is Disabled. Returning empty dict."
    )
    self.save_data_for_next_step(dict())
```

Chapter

5

ipaascore.BaseStep class

Overview

This chapter describes functions included in the `ipaascore.BaseStep` class. You can use these functions to define the logic in a step.

This chapter covers the following topics:

<code>get_app_variable</code>	63
<code>get_available_previous_step_input_positions</code>	64
<code>get_data_from_step_by_name</code>	65
<code>get_data_from_step_by_order</code>	66
<code>get_name</code>	67
<code>get_parameter</code>	68
<code>get_parameter_from_previous_step</code>	69
<code>join_previous_step_data</code>	70
<code>new_step_parameter</code>	72
<code>save_data_for_next_step</code>	73
<code>validate_parameter_values</code>	74

get_app_variable

Description

Retrieve the value of an application variable.

Syntax

```
get_app_variable(name)
```

Parameters

variable_name. The name of the application variable that you want to retrieve the value for.

Return

The value of the application variable.

Example

Suppose we defined this application variable in the integration application:

```
"app_variables": [  
  {  
    "name": "s11_hostname",  
    "description": "The SL1 hostname to participate in the sync",  
    "sample_value": "10.2.253.115",  
    "default_value": null,  
    "required": true,  
    "value": 10.64.68.25  
  },  
]
```

Suppose this integration application calls the step "sync_SL1_data".

In the step "sync_SL1_data", we could use the following function to resolve the value of "s11 hostname":

```
hostname = self.get_app_variable("s11_hostname")
```

The value of **hostname** would be "10.64.68.25".

get_available_previous_step_input_positions

Description

Retrieves the list of steps in the integration application along with the position of the step (the order that the step was run in the integration application. Position "0" (zero) is reserved for the current step's arguments.

Syntax

```
get_available_previous_step_input_positions()
```

Return

Returns a list of tuples. Each tuple includes a step name and the step's position.

get_data_from_step_by_name

Description

Retrieves data saved from a previous step.

NOTE: To retrieve data from a previous step, that previous step must save the data use the [save_data_for_next_step](#) function and the integration application must specify that the data from the previous step should be passed to the current step using the [output_to](#) parameter.

NOTE: Although the [get_data_from_step_by_name](#) function is simple to use, it does not allow you to write a generic, reusable step, because the step name will be hardcoded in the function. The functions [join_previous_step_data](#) or [get_data_from_step_by_order](#) allow you to create a more generic, reusable step.

Syntax

```
get_data_from_step_by_name(step_name)
```

Parameters

step_name. The name of a previous step in the integration application.

Return

The data that was saved by the previous step.

Example

The following is an example of the [get_data_from_step_by_name](#) function:

```
em7_data = self.get_data_from_step_by_name('FetchDevicesFromEM7')
snow_data = self.get_data_from_step_by_name('FetchDevicesFromSnow')
```

get_data_from_step_by_order

Description

This function retrieves data from a step based on the position of the step in the integration application.

NOTE: To retrieve data from a previous step, that previous step must save the data use the [save_data_for_next_step](#) function and the integration application must specify that the data from the previous step should be passed to the current step using the [output_to](#) parameter.

Syntax

```
get_data_from_step_by_order(pos)
```

Parameters

position. The position of the step (order that the step was run) in the integration application. Position values start at "0" (zero) .

Return

The data that was saved by the previous step.

Exception

DataNotAvailableException

Example

- Suppose your integration application has four steps: stepA, stepB, stepC, and stepD
- Suppose stepA was run first (position "0") and includes the parameter "output_to":["stepD"]
- Suppose stepB was run second (position "1") and includes the parameter "output_to":["stepD"]
- Suppose stepC was run third (position "2") and includes the parameter "output_to":["stepD"]
- Suppose stepD was run fourth
- If the current step is stepD, and stepD needs the data from stepC, you could use the following:

```
data_from_stepC = self.get_data_from_step_by_order(2)
```

get_name

Description

Returns the name of the current step.

Syntax

```
get_name()
```

Return

The name of the current step.

get_parameter

Description

Retrieves a parameter value.

Syntax

```
get_parameter(param_name, lookup_data=None)
```

Parameters

param_name. The name of the parameter that you want to retrieve the value for.

lookup_data. An optional dictionary that can provide a reference for additional variable substitutions.

Return

Value of the requested parameter.

Example

For example, suppose we defined this parameter in the step named "GETgoogle.com":

```
self.new_step_parameter(name=prefix_url, description="used with relative_url to create  
the full URL.", sample_value="http://10.2.11.253", default_value=None, required=True)
```

Suppose in the integration application that calls "GETgoogle.com", we specified:

```
"steps": [  
  {  
    "file": "QueryREST",  
    "method": "GET",  
    "name": "GETgoogle.com",  
    "output_to": ["next_step"],  
    "prefix_url": "http://google.com"  
  }  
],
```

Suppose we use the **get_parameter** function in the step "GETgoogle.com" to retrieve the value of the "prefix_url" parameter:

```
build_url_1 = self.get_parameter("prefix_url")
```

The value of **build_url_1** would be "http://google.com".

get_parameter_from_previous_step

Description

Retrieves a parameter value from a previous step.

Syntax

```
get_parameter_from_previous_step(parameter_name, step_name)
```

Parameters

param_name. The name of the parameter that you want to retrieve the value for.

step_name. the step from which you want to retrieve a parameter value.

Return

The value of the parameter.

join_previous_step_data

Description

This function retrieves data from one or more previous steps in the integration application.

NOTE: To retrieve data from a previous step, that previous step must save the data use the [save_data_for_next_step](#) function and the integration application must specify that the data from the previous step should be passed to the current step using the [output_to](#) parameter.

If you are expecting similar data from multiple steps, or expecting data from only a single step, the [join_previous_step_data](#) function is the best choice.

The [join_previous_step_data](#) function gathers all data from all steps that included the [save_data_for_next_step](#) function and also include the [output_to](#) parameter in the integration application. By default, this function returns the joined set of all data that is passed to the current step. You can also specify a list of previous steps from which to join data.

The retrieved data must be of the same type. The data is then combined into a list or a dictionary. If the data types are not the same, then the function will raise an exception

Syntax

```
join_previous_step_data(data_from)
```

Parameters

data_from. An optional argument that specifies the steps. For example, if you wanted to join only the data from stepA and stepD, you could specify

```
self.join_previous_step_data(["stepA", "stepD"]),
```

Return

Combined data structure, either a list or a dictionary, of all the retrieved step data.

Example

The following is an example of the `join_previous_step_data` function in the QueryREST step:

```
def query_with_url_generated_from_input(self):
    """
    Iterates over data from previous steps and generates a relative url for each. Then
    executes that command
    :return:
    """
    count = 0
    input_data = self.join_previous_step_data()
    payload = self.get_parameter(PAYLOAD)
    if type(input_data) is list:
        for input_d in input_data:
            relative_url = self.get_parameter(RELATIVE_URL, input_d)
            self.process_REST_command(payload, relative_url)
            count += 1
    elif type(input_data) is dict:
        relative_url = self.get_parameter(RELATIVE_URL, input_data)
        self.process_REST_command(payload, relative_url)
        count += 1
    else:
        raise NotImplementedError("Data type: {} is not currently supported for generating
        relative urls form data".format(type(input_data)))
```

new_step_parameter

Description

Define a parameter for the step. The integration application will provide the parameter values during runtime. If a parameter is defined as required, the step will fail with an exception if the parameter is not provided.

Syntax

```
new_step_parameter(name=" ", description=" ", sample_value=" ", default_value=None, required=False)
```

Parameters

name. The name of the parameter. This value will be used to create a name:value tuple in the integration application file (in JSON).

description. A description of the step parameter.

sample_value. A sample value of the required data type or schema.

default_value. If no value is specified for this parameter, use the default value. Can be any combination of alphanumeric characters. To prevent a default value, specify "None".

required. Specifies whether this parameter is required by the step. The possible values are "True" or "False".

Example

Here is an example from the QueryREST step:

```
self.new_step_parameter(name=PREFIX_URL, description="used with relative_url to create the full URL.", sample_value="http://10.2.11.253", default_value=None, required=True)
```

save_data_for_next_step

Description

This function saves an object (usually a variable) and makes the data available to another step. The object must be of a data type that can be pickled by Python.

Syntax

```
save_data_for_next_step(data_to_save)
```

Parameters

data_to_save. A variable that contains the data.

NOTE: The *data_to_save* object must be of a data type that can be pickled by Python: None, True and False, integers, long integers, floating point numbers, complex numbers, normal strings, unicode strings, tuples, lists, set, and dictionaries.

Example

The following is an example of the **save_data_for_next_step** function:

```
save_data = {'key': 'value'}  
self.save_data_for_next_step(save_data)
```

The integration application must then specify that the data from the current step should be passed to one or more subsequent step, using the **output_to** parameter. For details, see the section about [using integration applications to transfer data between steps](#).

validate_parameter_values

Description

Validates the parameter values provided for an integration application. For example, the `validate_parameter_value` function will raise an error if the user failed to provide a required parameter.

Syntax

```
validate_parameter_values()
```

Tools for the Integration Service

Overview

The Integration Service includes the following tools to help you manage content:

- **iscli**. The Integration Service includes a command line tool called **iscli**. When you install Integration Service, **iscli** is automatically installed. The **iscli** allows you to upload integration applications, steps, and configurations.
- **IS API**. The Integration Service includes an API. When you install the Integration Service, the API is available.

This chapter covers the following topics:

iscli	76
IS API	78

iscli

The Integration Service system includes a command line tool called **iscli**. When you install the Integration Service system, **iscli** is automatically installed. The iscli allows you to upload integration applications, steps, and configurations.

To access the iscli tool:

1. Either go to the console of the Integration Service system or use SSH to access the server.
2. Log in as **isadmin** with the appropriate password.
3. Type the following at the command line:

```
iscli -h
```

Syntax

- To upload a step:

```
iscli -u -s -f path_and_name_of_step_file.py -H hostname_or_IP_address_of_integration_service_system -P port_number_of_http_on_integration_service_system -U user_name -p password
```

- To upload an application:

```
iscli -u -a -f path_and_name_of_application_file.json -H hostname_or_IP_address_of_integration_service_system -P port_number_of_http_on_integration_service_system -U user_name -p password
```

- To upload a configuration:

```
iscli -u -c -f path_and_name_of_configuration_file.json -H hostname_or_IP_address_of_integration_service_system -P port_number_of_http_on_integration_service_system -U user_name -p password
```

List of Arguments

The following is a list of arguments you can use in the iscli tool:

- **-h or --help**. Help message.
- **-H or --host**. Hostname of content store. If left blank, default from config is used.
- **-P or --port**. Port of content store. If left blank, default from config is used.
- **-U or --username**. Username used to access content store. If left blank, default is used.
- **-p or --password**. Password used to access content store. If left blank, default is used.
- **-u or --upload**. Upload data to content store. Requires **-f** tag and a version to be used.

- **-f or --fileLocation.** File path of Synchronization PowerPacks, integration applications, steps, and configuration objects to upload. Note that while you can bulk upload integration applications, steps, and configuration objects, you cannot bulk upload Synchronization PowerPacks.
- **-c or --configuration.**
- **-a or --application.** Use the iscli tool to interact with an application.
- **-s or --step.** Use the iscli tool to interact with a step.
- **-i or --insecure.** Upload with an insecure (http) connection.
- **-k or --syncpack.** Use the iscli tool to interact with a Synchronization PowerPack.
- **-A or --activate.** Activate a Synchronization PowerPack. Requires **-n**, **-v** tag. **-F** is optional to force the activation and the tag **-l** to install the Synchronization PowerPack.
- **-l or --install.** Install a Synchronization PowerPack. Requires **-n**, **-v** tag. **-F** is optional to force the installation.
- **-n or --name.** Name of the Synchronization PowerPack that will be activated.
- **-V or --version.** Version of the Synchronization PowerPack that will be activated.
- **-F or --force.** Force a Synchronization PowerPack activation or installation.
- **-O or --overwrite.** Specify if non-Synchronization PowerPack content will be overwritten.
- **-l or --license.** Use the CLI to license your IS system.
- **-e or --email.** Licensing: Customer Email.
- **-C or --customer.** Licensing: Customer Name.
- **-o or --output.** Licensing: Path to output license file.

You will use the following commands most often:

- **Upload an application file.** `iscli -uaf <app-file>`
- **Upload a configuration file.** `iscli -ucf <config-fil>`
- **Upload a step file.** `iscli -usf <step-file>`

Example

```
iscli -u -s -f path_and_name_of_step_file.py -H hostname_or_IP_address_of_integration_service_system -P port_number_of_http_on_integration_service_system -U user_name -p password
```

where:

- *name_of_step_file* is the full pathname for the step.
- *hostname_or_IP_address_of_integration_service_system* is the hostname or IP address of the Integration Service system.
- *port_number_of_http_on_integration_service_system* is the port number to access the Integration Service system.
- *user_name* is the user name you use to log in to the Integration Service system.
- *password* is password you use to log in to the Integration Service system.

IS API

The Integration Service includes an API. When you install the Integration Service, the API is available.

Viewing API Documentation

To view the full documentation for the IS API:

1. From the Integration Service system, copy the file `/opt/iservices/scripts/swagger.yml` to your local computer.
2. Open a browser session and go to editor.swagger.io.
3. In the Swagger Editor, open the **File** menu, select **Import File**, and import the file `swagger.yml`. The right pane in the Swagger Editor displays the IS API documentation.

Available Endpoints

POST

`/applications`. Add a new application or overwrite an existing application.

`/applications/{appName}/run`. Run a single application by name with saved or provided configurations.

`/applications/run`. Run a single application by name. For more information, see [Querying for the State of an Integration Application](#).

`/configurations`. Add a new configuration or overwrite an existing configuration.

`/license`. Add license data for this Integration Service system.

`/roles/owner`. Add a new owner assigned a specific role.

/steps. Add a new step or overwrite an existing step.

/steps/run. Run a single step by name.

/schedule. Add a new scheduled application integration.

/syncpacks/{syncpackName}/install. Install a specific Synchronization PowerPack version by name.

/tasks/{taskId}/replay. Replay a specific integration application. Replayed integration applications run with the same application variables, configuration, and queue as the originally executed application.

/tasks/{taskId}/revoke. Revoke or terminate a specific task or integration application. If an application ID is provided, all tasks associated with that integration application are be revoked.

Querying for the State of an Integration Application

When triggering an integration application from the **applications/run** endpoint, you can query for the state of that integration application in two ways:

1. **Asynchronously.** When you POST a run of an integration application to **/applications/run**, the response is a integration status with a Task ID, such as: *isap-23233-df2f24-etc*. At any time, you can query for the current state of that task from the endpoint **/api/v1/tasks/isap-23233-df2f24-etc**. The response includes all of the steps run by the integration application, along with the status of the steps, and URL links to additional info, such as logs for each step.
2. **Synchronously.** When you POST a run of an integration application, you can tell the Integration Service to wait responding until the integration application is complete by adding the **wait** argument. For example, **/api/v1/applications/run?wait=20** will wait for 20 seconds before responding. The maximum wait time is 30 seconds. When the integration application completes, or 30 seconds has passed, the API returns the current status of the integration run. This process works the same as if you had manually queried **/api/v1/tasks/isapp-w2ef2f2f**. Please note that while the API is waiting for your integration application to complete, you are holding on to a thread. If you have multiple integration applications that run for a long period of time, do not use a synchronous query unless you have no other option. ScienceLogic recommends using an *asynchronous* query whenever possible.

GET

/about. Retrieve version information about the packages used by this Integration Service system.

/applications. Retrieve a list of all available applications on this Integration Service system.

/applications/{appName}. Retrieve a specific application.

/applications/{appName}/logs. Retrieve the logs for the specified application.

/cache/{cache_id}. Retrieve a specific cache to gather information about the user interface and the integration applications.

/configurations. Retrieve a list of all configurations on this Integration Service system.

/configurations/{configName}. Retrieve a specific configuration.

/license. Retrieve license data for this Integration Service system.

/reports. Retrieve a list of paginated reports.

/reports/{reportId}. Retrieve a specific report by ID.

/roles. Retrieve a list of available roles on this Integration Service system.

/roles/owner. Retrieve a list of roles assigned to owners on this Integration Service system.

/roles/owner/{owner}. Retrieve the role assigned to a specific owner.

/schedule. Retrieve a list of all scheduled application integrations on this Integration Service system.

/steps. Retrieve a list of all steps on this Integration Service system.

/steps/{stepName}. Retrieve a specific step.

/syncpacks. Retrieve a list of all Synchronization PowerPacks on this Integration Service system.

/syncpacks/{synpackName}. Retrieve the full details about a specific Synchronization PowerPack.

/api/v1/syncpacks?only_installed=true. Retrieve a list of only the installed Synchronization PowerPacks on this system.

/api/v1/syncpacks?only_activated=true. Retrieve a list of only the activated Synchronization PowerPacks on this system.

/tasks/{taskId}. Retrieve a specific task.

REST

/tasks. Terminate all running tasks.

/tasks/{taskId}. Terminate a specific running task.

DELETE

/applications/{appName}. Delete an integration application by name.

/cache/{cache_id}. Delete a cache entry by name.

/configurations/{configName}. Delete a configuration by name.

/license. Delete license data for this Integration Service system.

/roles/owner. Delete a specific owner role.

/schedule. Delete a scheduled application integration by ID.

/reports/{appName}. Delete a specific report by name.

/reports/{reportId}. Delete a specific report by report ID.

/steps/{stepName}. Delete a specific step by name.

/syncpacks/{spName}. Delete a specific Synchronization PowerPack by name.

© 2003 - 2020, ScienceLogic, Inc.

All rights reserved.

LIMITATION OF LIABILITY AND GENERAL DISCLAIMER

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 on this Site, information on this Site 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 in this Site at any time without notice.

Copyrights and Trademarks

ScienceLogic, the ScienceLogic logo, and EM7 are trademarks of ScienceLogic, Inc. in the United States, other countries, or both.

Below is a list of trademarks and service marks that should be credited to ScienceLogic, Inc. The ® and ™ symbols reflect the trademark registration status in the U.S. Patent and Trademark Office and may not be appropriate for materials to be distributed outside the United States.

- ScienceLogic™
- EM7™ and em7™
- Simplify IT™
- Dynamic Application™
- Relational Infrastructure Management™

The absence of a product or service name, slogan or logo from this list does not constitute a waiver of ScienceLogic's trademark or other intellectual property rights concerning that name, slogan, or logo.

Please note that laws concerning use of trademarks or product names vary by country. Always consult a local attorney for additional guidance.

Other

If any provision of this agreement shall be unlawful, void, or for any reason unenforceable, then that provision shall be deemed severable from this agreement and shall not affect the validity and enforceability of any remaining provisions. This is the entire agreement between the parties relating to the matters contained herein.

In the U.S. and other jurisdictions, trademark owners have a duty to police the use of their marks. Therefore, if you become aware of any improper use of ScienceLogic Trademarks, including infringement or counterfeiting by third parties, report them to Science Logic's legal department immediately. Report as much detail as possible about the misuse, including the name of the party, contact information, and copies or photographs of the potential misuse to: legal@sciencelogic.com



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

International: +1-703-354-1010