Informatica

Informatica® INFACore
November 2023

JupyterLab Extension for INFACore

Informatica INFACore JupyterLab Extension for INFACore
November 2023

Publication Date: 2023-11-07

Table of Contents

Preface

Read *JupyterLab Extension for INFACore* to get familiar with INFACore and understand how to use INFACore to manage data for your business from your JupyterLab development environment.

Informatica Resources

Informatica provides you with a range of product resources through the Informatica Network and other online portals. Use the resources to get the most from your Informatica products and solutions and to learn from other Informatica users and subject matter experts.

Informatica Documentation

Use the Informatica Documentation Portal to explore an extensive library of documentation for current and recent product releases. To explore the Documentation Portal, visit https://docs.informatica.com.

If you have questions, comments, or ideas about the product documentation, contact the Informatica Documentation team at infa_documentation@informatica.com.

Informatica Intelligent Cloud Services web site

You can access the Informatica Intelligent Cloud Services web site at http://www.informatica.com/cloud. This site contains information about Informatica Cloud integration services.

Informatica Intelligent Cloud Services Communities

Use the Informatica Intelligent Cloud Services Community to discuss and resolve technical issues. You can also find technical tips, documentation updates, and answers to frequently asked questions.

Access the Informatica Intelligent Cloud Services Community at:

https://network.informatica.com/community/informatica-network/products/cloud-integration

Developers can learn more and share tips at the Cloud Developer community:

https://network.informatica.com/community/informatica-network/products/cloud-integration/cloud-developers

Informatica Intelligent Cloud Services Marketplace

Visit the Informatica Marketplace to try and buy Data Integration Connectors, templates, and mapplets:

https://marketplace.informatica.com/

Data Integration connector documentation

You can access documentation for Data Integration Connectors at the Documentation Portal. To explore the Documentation Portal, visit https://docs.informatica.com.

Informatica Knowledge Base

Use the Informatica Knowledge Base to find product resources such as how-to articles, best practices, video tutorials, and answers to frequently asked questions.

To search the Knowledge Base, visit https://search.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Intelligent Cloud Services Trust Center

The Informatica Intelligent Cloud Services Trust Center provides information about Informatica security policies and real-time system availability.

You can access the trust center at https://www.informatica.com/trust-center.html.

Subscribe to the Informatica Intelligent Cloud Services Trust Center to receive upgrade, maintenance, and incident notifications. The Informatica Intelligent Cloud Services Status page displays the production status of all the Informatica cloud products. All maintenance updates are posted to this page, and during an outage, it will have the most current information. To ensure you are notified of updates and outages, you can subscribe to receive updates for a single component or all Informatica Intelligent Cloud Services components. Subscribing to all components is the best way to be certain you never miss an update.

To subscribe, go to https://status.informatica.com/ and click **SUBSCRIBE TO UPDATES**. You can then choose to receive notifications sent as emails, SMS text messages, webhooks, RSS feeds, or any combination of the four.

Informatica Global Customer Support

You can contact a Customer Support Center by telephone or online.

For online support, click **Submit Support Request** in Informatica Intelligent Cloud Services. You can also use Online Support to log a case. Online Support requires a login. You can request a login at https://network.informatica.com/welcome.

The telephone numbers for Informatica Global Customer Support are available from the Informatica web site at https://www.informatica.com/services-and-training/support-services/contact-us.html.

CHAPTER 1

Introducing JupyterLab extension for INFACore

If you are a developer, data scientist, data engineer, or machine learning engineer, and you prefer low-code data integration capabilities for your diverse data management use cases, you can plug in INFACore into your JupyterLab environment and take your data management projects from ideation to deployment, in a real quick and cost-effective manner.

The JupyterLab extension for INFACore is a JupyterLab-based development environment for INFACore. The extension provides you with a user interface for INFACore that helps you interact with INFACore from within JupyterLab. Use the INFACore SDK for Python to execute code to manage data in the INFACore instance in JupyterLab.

INFACore helps you experience the power and functionality of Informatica Intelligent Cloud Services from your JupyterLab environment while allowing you to retain complete control over the front-end experience of your application.

You can manage data from disparate sources directly from your JupyterLab environment. Use INFACore to extract source data, transform the data, and then load it to your target.

Your data science project might require you to use INFACore to perform the following tasks:

- Explore data that is already residing in your development environment.
- Access a data source from JupyterLab to either fetch data into JupyterLab or to write data from JupyterLab to the required data endpoint.
- Apply built-in functions to validate, standardize, and cleanse your data.
- Use Intelligent Structure Models to parse semi-structured and unstructured data.

INFACore connects to your data source and executes the operations based on the configurations you apply.

Got a question?

Have a quick question on INFACore? Check out these questions and answers.

Do I need to use the JupyterLab extension for INFACore to use the INFACore Python SDK?

No, the INFACore Python SDK works independently of the JupyterLab extension for INFACore. You can call the INFACore functions using the SDK directly within any Python development environment after importing the SDK dependent libraries and packages.

However, if you are using the JupyterLab extension for INFACore, you also need the INFACore Python SDK. The extension provides you with a user interface with interactive controls that you can use to

perform simple tasks such as to log into INFACore and perform basic functions, which helps you speed up the development process. The Python SDK works alongside the extension to invoke the INFACore methods and classes.

What programming language does the INFACore SDK support?

INFACore includes the SDK for the Python programming language with which you can leverage the functionalities of INFACore.

For more information about the INFACore Python SDK, see the INFACore SDK Reference for Python.

What operating system does the JupyterLab extension for INFACore support?

INFACore extension for JupyterLab supports the Windows and Linux operating systems.

What browsers can I use?

You can use the latest versions of the following browsers:

- Firefox
- Chrome
- Safari

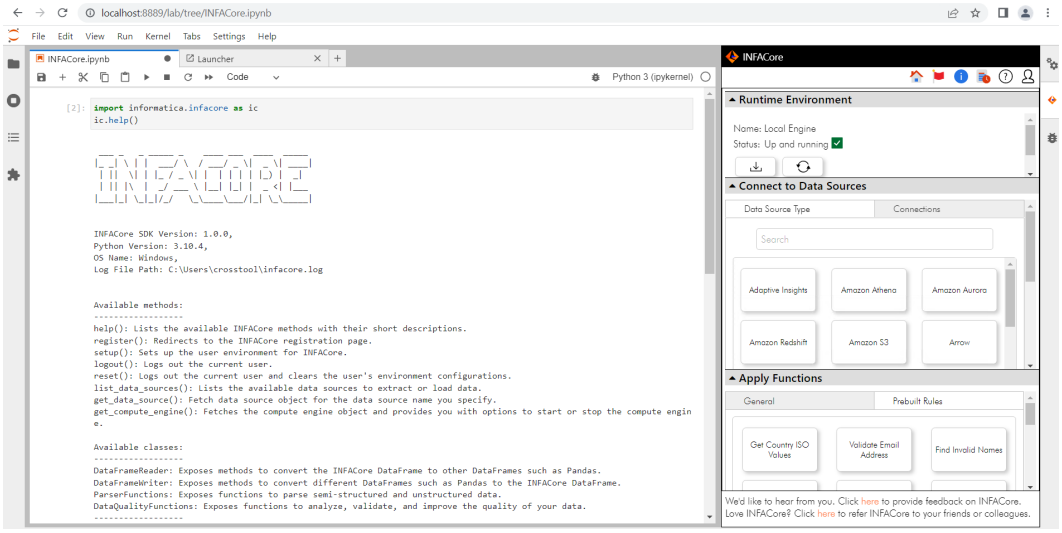
Can I use INFACore to access any data source?

You can connect to around 70 data sources. See the Connections documentation to view the list of data sources that you can access using INFACore.

Let's get started

You can create your first job using INFACore in just a few steps!

When you log in to INFACore from JupyterLab, the INFACore Home page appears in JupyterLab, as shown in the following image:



On the left-hand side, you can see the JupyterLab work area where you can programmatically invoke the INFACore Python SDK. On the right-hand side, the JupyterLab interface for INFACore provides you with the options to perform basic operations to get you started.

You can perform the following tasks from the INFACore user interface:

1. Set up the runtime environment.

   Before you can connect to a data source or to apply functions to a data source, you need to install the runtime environment. The runtime environment is the execution platform for running all the INFACore jobs.

2. Connect to your data source.

   The data source is the endpoint from where you want to read from or write data. Select the data source to which you want to connect and then create a connection to connect to the data source.

3. Apply functions to your data.

   Functions represent the operations that you want to perform on the data.

To invoke the INFACore Python SDK for various operations, see the *INFACore SDK Reference for Python*.

After you create your first job, you can also:

- Monitor the activity of your jobs and operations in the **Activity logs** page.
- Download the log files for a job to the agent machine.
- Use the in-built forms to refer INFACore to colleagues, write a feedback about INFACore, or to submit requests. In the **Connect to Data Sources** section, you can request for new connections to access data source endpoints from INFACore. In the **Apply Functions** section, you can also submit a request for new functions, with the purpose defined.

End-to-end quick tour

You are a data scientist and you want to explore the JupyterLab extension for INFACore to manage data from your JupyterLab environment.

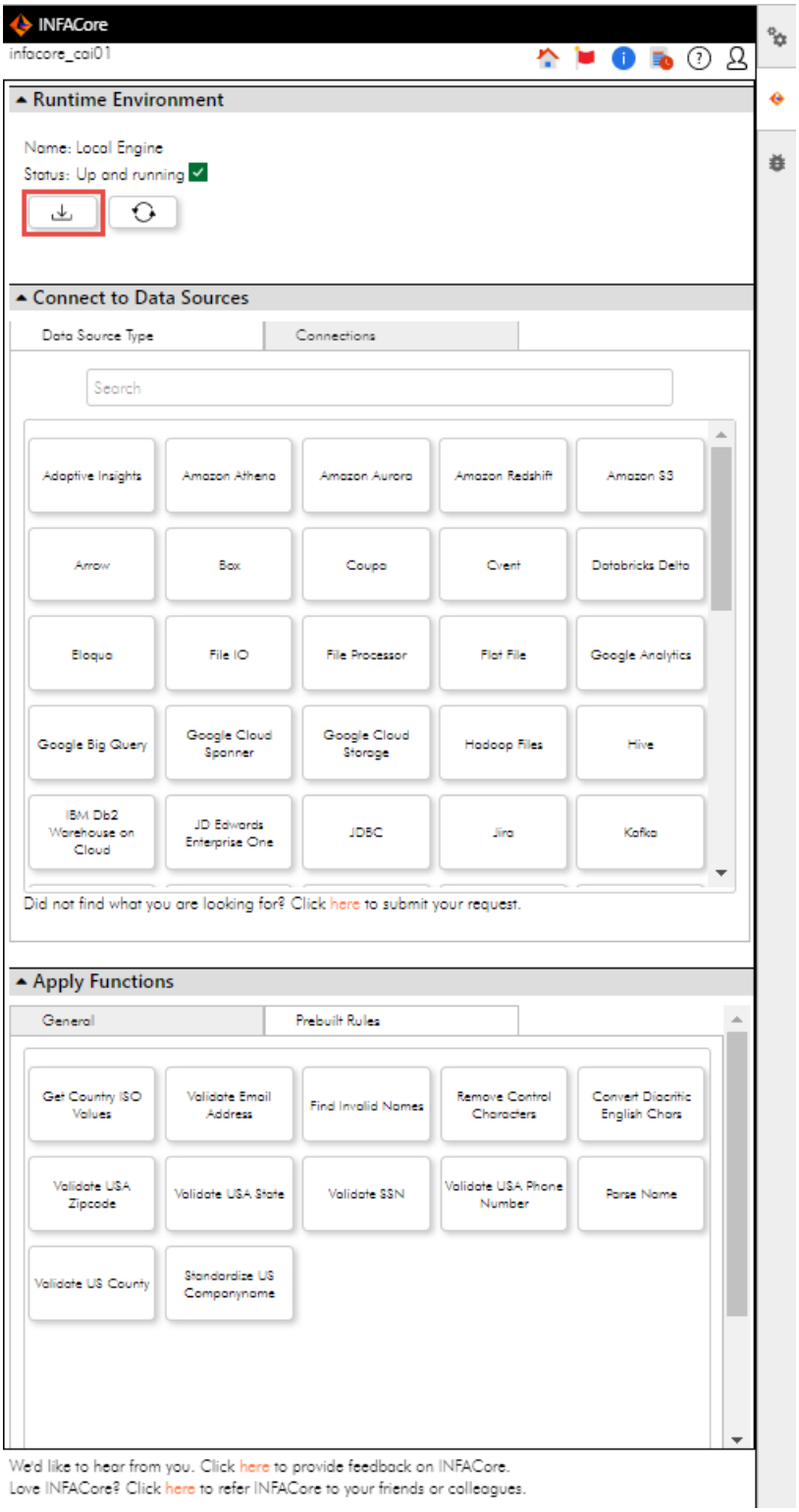
To get started, perform the following tasks after you log in to INFACore:

Step 1. Set up the runtime environment

The runtime environment is the execution platform that runs the INFACore jobs.

First, let's install the agent on the machine that hosts your development environment.

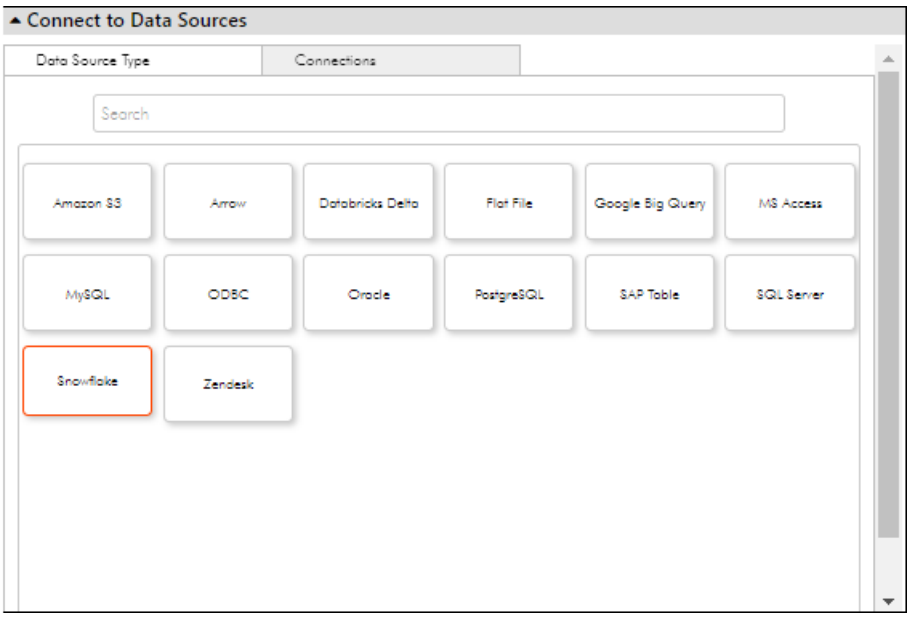
1. In the **Runtime Environment** section, click to install the agent on your machine.
The runtime environment downloads an agent locally on your machine and the status displays as up and running, as shown in the following image:



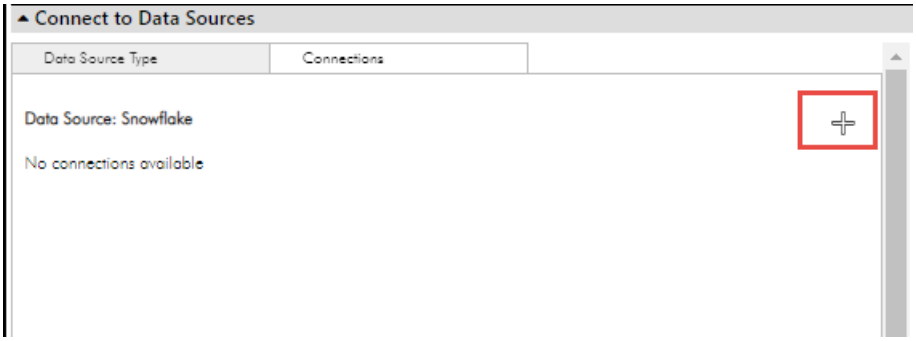
Step 2. Connect to the data source

First, select and configure the data source to which you want to connect. You can select an existing connection or create a new connection to connect to your data source.

1. In the **Connect to Data Sources** section, click the **Data Source Type** tab, and then select the data source from the list.
   You can also search for the data source from the list.

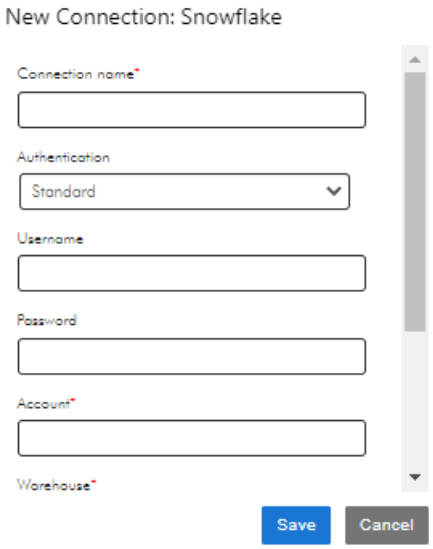


2. On the **Connections** tab, select an existing connection for the data source from the list, or create a new connection to the data source.
3. To create a new connection, click the + icon, and then specify the details for the data source that you want to connect to.



   For example, if you configure a Snowflake connection, enter a name for the connection, select the authentication method, and enter the Snowflake account details.

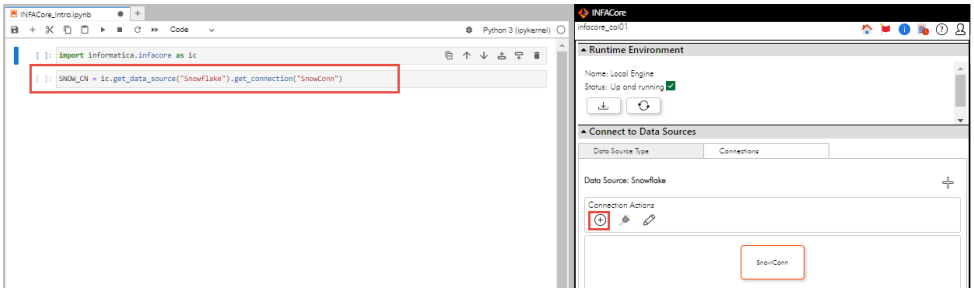
The following image shows the properties for a Snowflake connection:



When you create and save a new connection, that connection displays in the connection list.

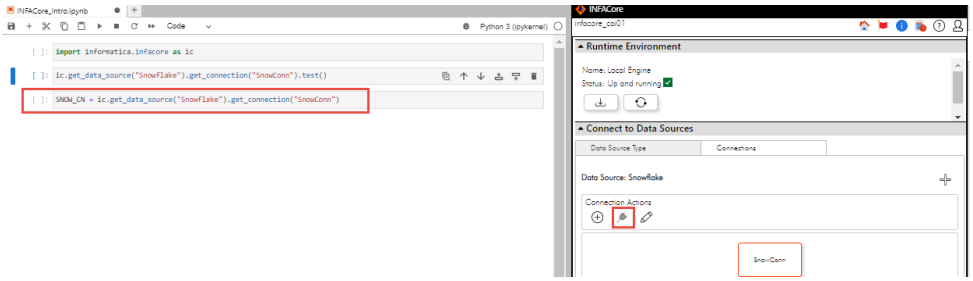
4. Select the required connection and perform one of the following actions:

   a. To add the connection to the code cell, select the ⊕ icon, and provide a variable name for the connection to display in the Python code.
   The selected connection is added to the Python code.



   b. To edit the connection, select the icon, edit the connection details, and save the connection.

   c. To test the connection, select the icon.
   The Python code for testing the selected connection displays.



   Run the code to test if you can connect to Snowflake.

Step 3. Explore the data

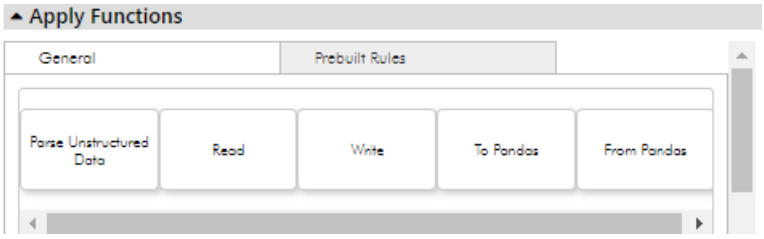
After you configure the data source, you can configure functions on your data to perform the following operations:

Read, write, convert to and from the pandas dataFrame.

Parse unstructured or semi-structured data.

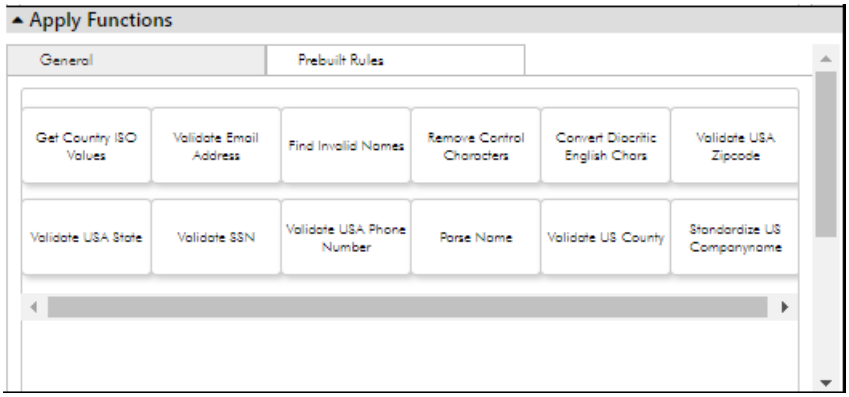
Apply prebuilt rules to analyze the data.

1. On the **General** tab, you can perform the following operations, and click **Submit**:



   a. To read from or to write data, specify the data source, connection, and the data object name.
   b. To convert to or from the Pandas DataFrame, specify a variable name for the Pandas DataFrame.
   c. To apply the parser function on unstructured or semi-structured data, provide a name for the data source, and specify the paths to the sample schema and the input file for the data to which you want to apply the parser function.

2. On the **Prebuilt Rules** tab, select the required pre-built rule to apply to your data, perform the following tasks, and click **Submit**:



   a. Enter the variable name for the source.
   b. Specify the applicable column name for the field based on the rule you select.

That's it! When you run the code, INFACore performs the configured operations on the data. If you want to check your activity, you can see it on the **Activity Log** page.

To configure any of these operations, you can also directly invoke the INFACore Python SDK. For more information about configuring these operations using the INFACore SDK for python, see the "Read and write end-to-end example" in the "Quickstart" section in the *INFACore SDK Reference for Python*.

CHAPTER 2

Set up the runtime environment

The runtime environment is the execution platform that runs the INFACore jobs.

To set up the runtime environment, you need to install the agent. The installation downloads an agent, installs, and registers the agent to your organization. The agent is a lightweight program that runs all jobs and enables secure communication across the firewall between your organization and INFACore.

If you are a first-time user, you install the agent on the Windows or Linux machine locally on your machine that hosts your development environment.

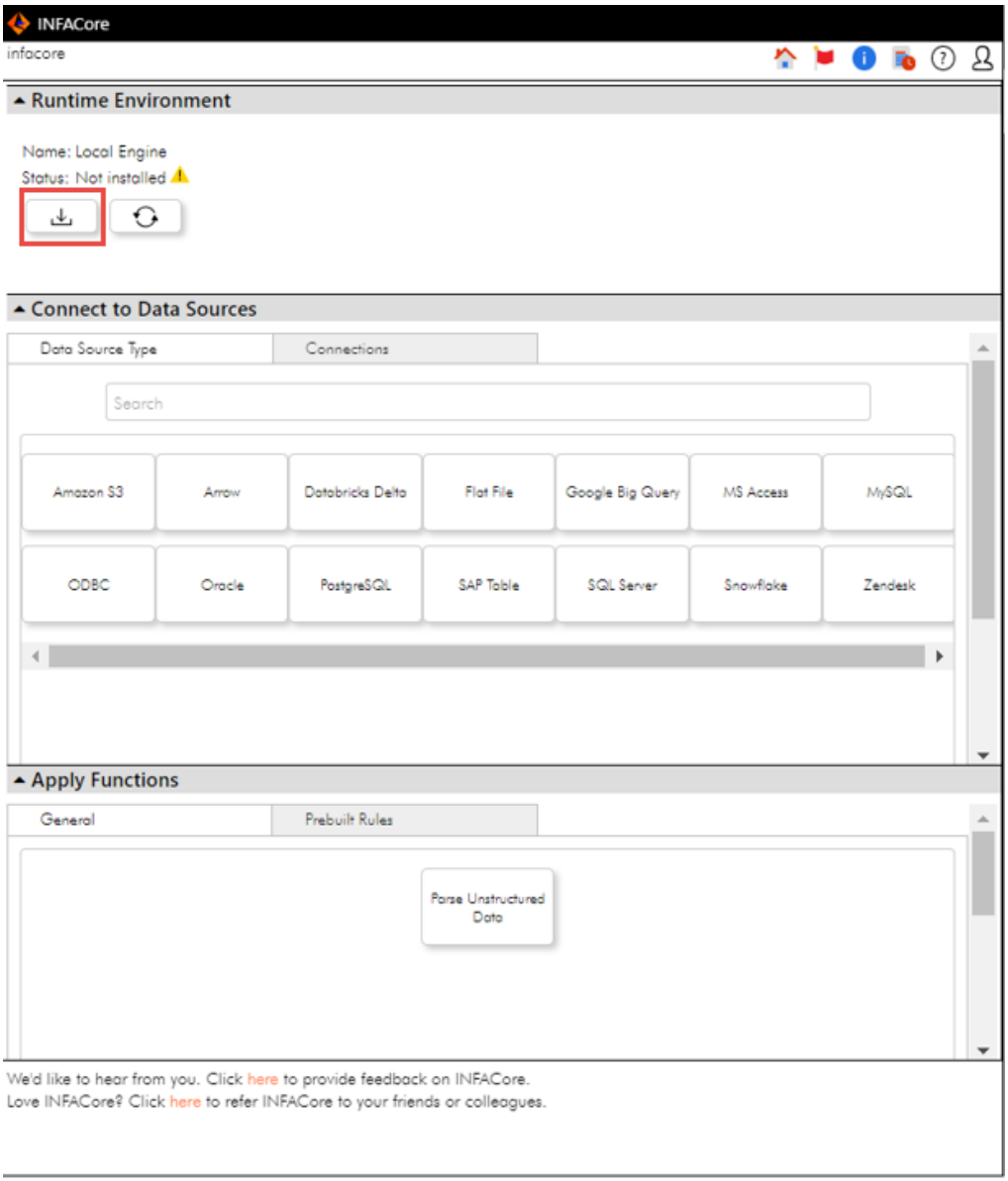
After you install, the status of the agent displays as up and running, and you can then continue to the next step. You can stop and restart the agent from the Informatica Cloud Secure Agent page.

Installing the agent

Install the agent from the **Runtime Environment** section. INFACore installs the agent locally on your machine.

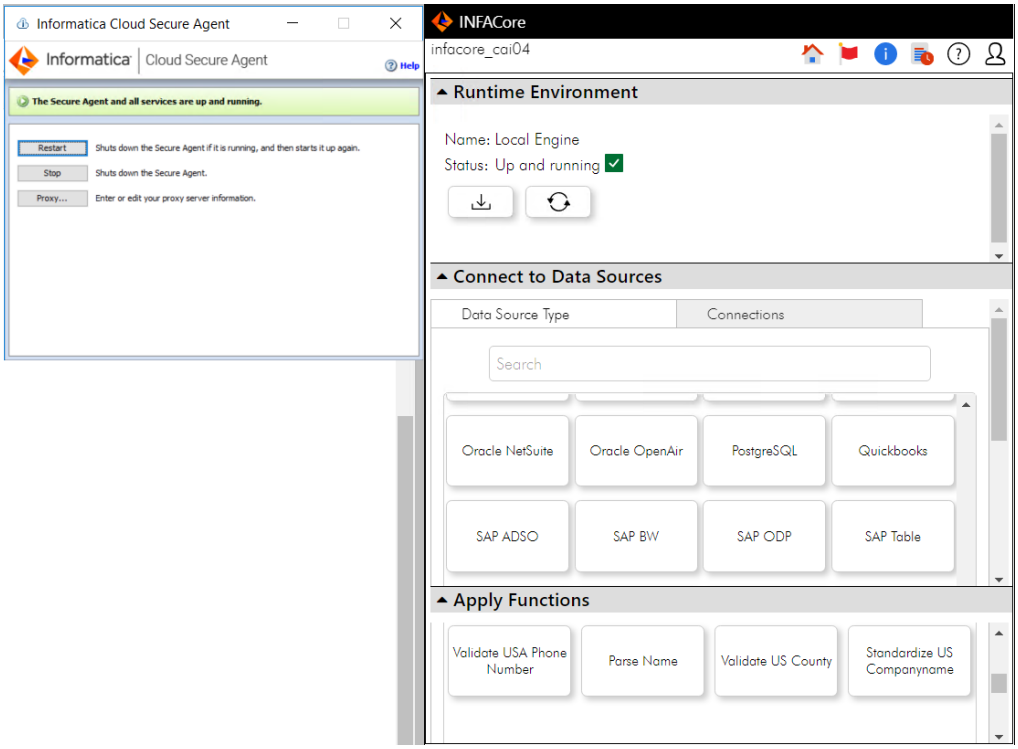
The installer also installs the Informatica Cloud Secure Agent Manager with options to stop or restart the agent.

1. On the **Home** page, click the icon to install the agent on your machine.



2. If prompted to allow your application to make changes to your device, enter your system user name and password to allow the program to install the agent.
3. When prompted to register the agent, there is no action required from you.

The engine downloads locally on your machine and the status of the engine displays as up and running, as shown in the following image:



4. You can restart or stop the agent from the Informatica Cloud Secure Agent page.

   If you install the agent on Windows, you can also stop and restart the agent from the Windows Services. On Linux, the agent runs as a process. You can use a shell command line to start and stop the agent.

   Note: The proxy configuration is not applicable for INFACore.

Uninstalling the agent

You can uninstall the Secure Agent. You might uninstall the Secure Agent if you no longer want to run the Secure Agent on the machine or if you want to reinstall the Secure Agent.

Before you uninstall the Secure Agent, verify that no connection or task is configured to use it.

1. Click **Start** > **All Programs** > **Informatica Cloud Secure Agent** > **Uninstall Informatica Cloud Secure Agent**.

   The Secure Agent uninstaller launches.

2. Click **Uninstall**.
3. When the uninstall completes, click **Done**.
4. Delete any remaining files in the installation directory.

   After you uninstall the Secure Agent, delete all files and directories associated with the Secure Agent installation.

Note: Uninstalling the Secure Agent does not delete log files from the Secure Agent directory. If you want to reinstall a Secure Agent on the machine, you must delete all files and directories associated with the Secure

Agent installation or reinstallation will fail. If you want to save the log files, copy them to a different directory, and then delete the Secure Agent installation directory.

Guidelines for the runtime environment

After you install the runtime environment, you do not have to reinstall it the next time you log in. If you try to install again, a message displays informing you that the agent is already installed and whether you want to install again. You can click **No**.

If you want to re-install the agent, make sure that you do not perform the installation when the agent is up and running.

CHAPTER 3

Connect to the data source

To connect to a data source, specify the data source that you want to connect to and then create a connection to enable the exchange of data between your development environment and the source and target endpoints.

You can use INFACore to connect to a wide variety of data sources for data integration. Specify the data sources to create connections that provide access to data in cloud and on-premise applications, platforms, databases, and flat files.

Some data integration settings are connector-specific. For more information, see the help for the relevant data source.

Data source types

When you select a data source type, you'll need to create or select a connection to connect to the source or target endpoint. INFACore uses the connection information to read data from your source or load data to your target.

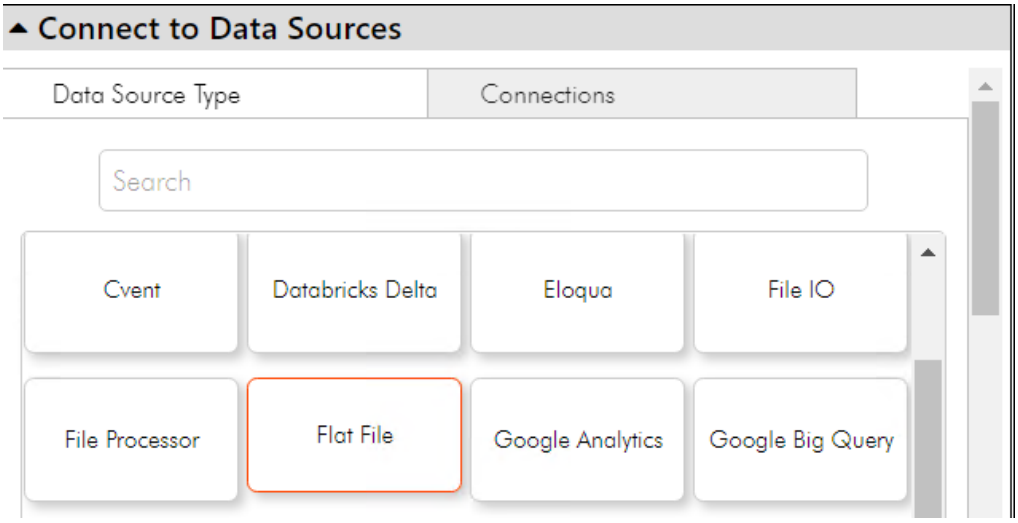
Create a connection in the **Connect to Data Sources** section in INFACore.

When you create a connection, you enter the connection properties. Connection properties vary based on the connection type. For example, for an Amazon S3 data source, you'll need to enter the access key, secret key, and other properties. For more information about the connection properties for different connection types, see *Connections*.

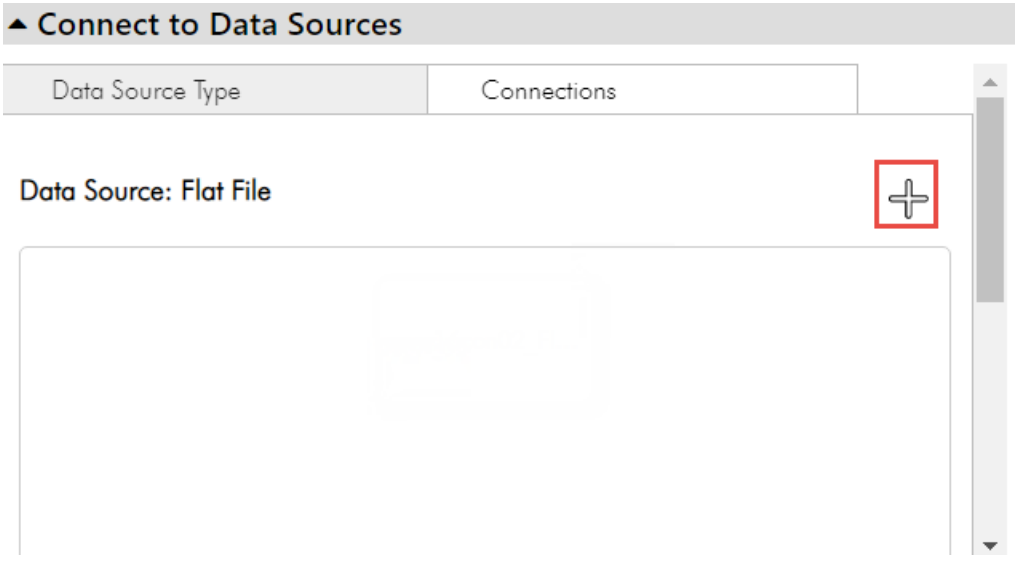
Configure a connection to the data source

Select a data source and configure a connection. Add code for the connection and then test the connection.

1. On the **Data Source Type** tab, select the data source from the list or search for a data source. You can search for the data source from the list.

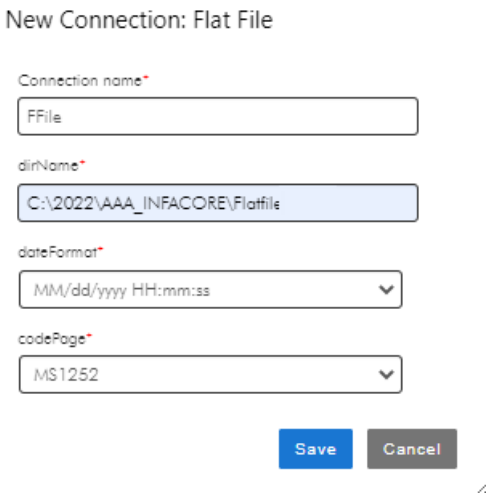


2. On the **Connections** tab, select an existing connection for the data source from the list, or create a new connection to the data source.
3. To create a new connection, click the + icon and configure the connection-specific properties for the data source that you want to connect to.



For example, if you configure a flat file connection, enter the directory where the files are stored, the date format for the date fields in the files, and the code page of the system that hosts the files.

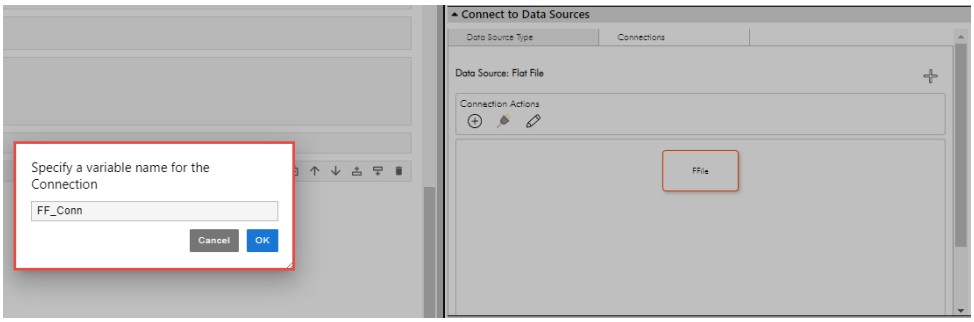
The following image shows the properties that you need to configure for a flat file connection:



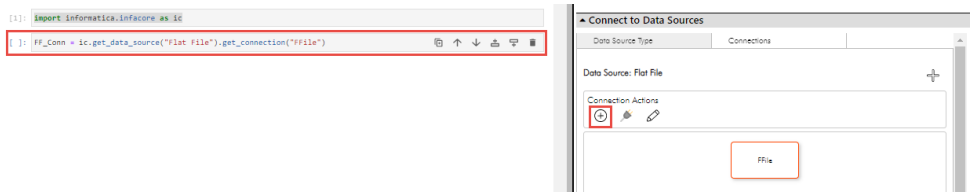
When you create and save a new connection, that connection displays in the connection list.

4. Select the required connection and perform one of the following actions:

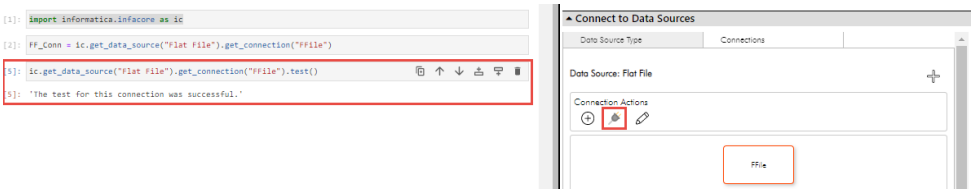
   a. To add the connection to the code cell, select the ⊕ icon, and provide a variable name for the connection to display in the Python code.



   The selected connection is added to the Python code.

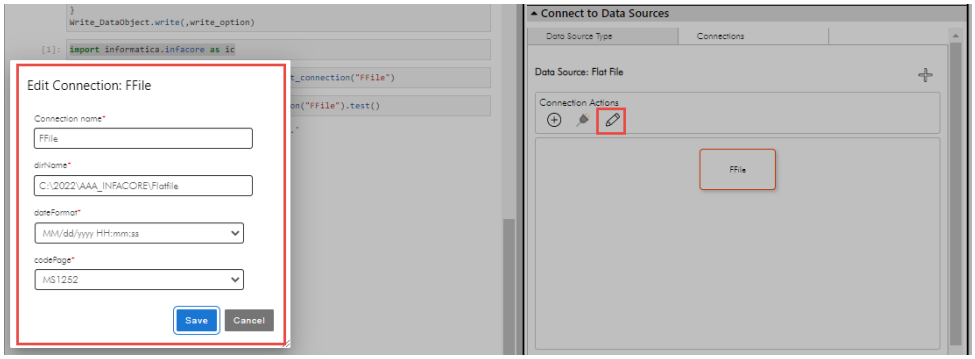


   b. To test the connection, select the icon.



   The Python code for testing the selected connection displays. When you run the code, the test connection displays as successful.

c. To edit the connection, select the icon, edit the connection details, and save the connection.



Create a data object

After you create a connection for a data source, you need to create a data object variable that you can use in INFACore operations.

To create the data object, call the `get_data_object` method in the Python SDK and provide the details of the data object instance.

See the following example code to create a flat file data object variable:

```
ff_do =
ic.get_data_source("FlatFile").get_connection("CN_FlatFile").get_data_object("salesinput.
csv")
```

In the code:

- `ff_do` is the data object variable name for the flat file.
- `get_data_source("Flat File")` is the method to get the flat file data source.
- `get_connection("CN_FlatFile")` is the method to get the flat file connection.
- `get_data_object("salesinput.csv")` is the method to create the data object instance.

You can use the configured data object variable when you apply prebuilt functions or other operations on the data object instance.

CHAPTER 4

Explore the data

You can apply functions on your data to help you analyze the content and structure of your data and transform your data based on your business requirements. You can improve the quality of your business data and create quality data sets.

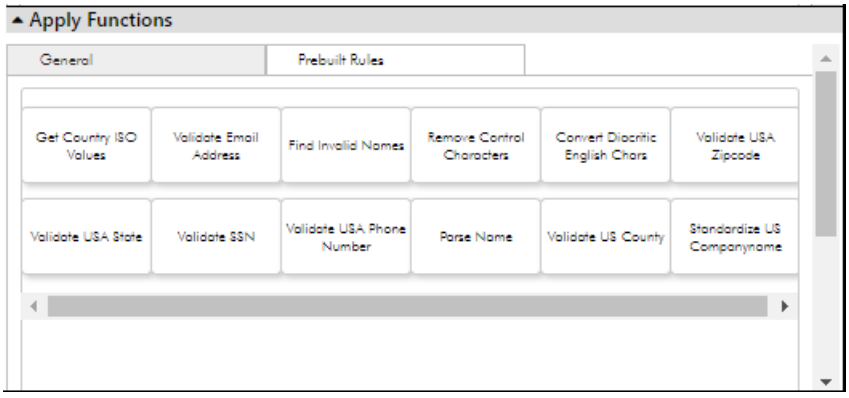
After you connect to a data source, you can perform the following operations on your data:

- Read from or write data to a data source end point.
- Choose to apply the Pandas functions to your data to represent the data in a Pandas DataFrame.
- If your data is converted to a Pandas DataFrame, you can convert it to the Informatica DataFrame before you write to the target.

  Note: The time that the to_pandas function takes to convert data to a Pandas DataFrame and for the from_pandas function to convert the data from the Pandas DataFrame to the Informatica DataFrame scales linearly with the size of the data that you are processing.
- Apply parser transformations to parse unstructured or semi-structured data.
- Apply a number of built-in rules to verify, standardize, and address issues in the source data.

The following image shows the **General** tab with the list of operations that you can perform on your data:

The following image shows the **Functions** tab with the list of data quality functions that you can apply on your data:



When you select a function and specify the data source and the column name for which you want to apply the function, the parameters are added to the code in your development environment and you can run the code to get the output you want.

Read data and convert to Pandas DataFrame

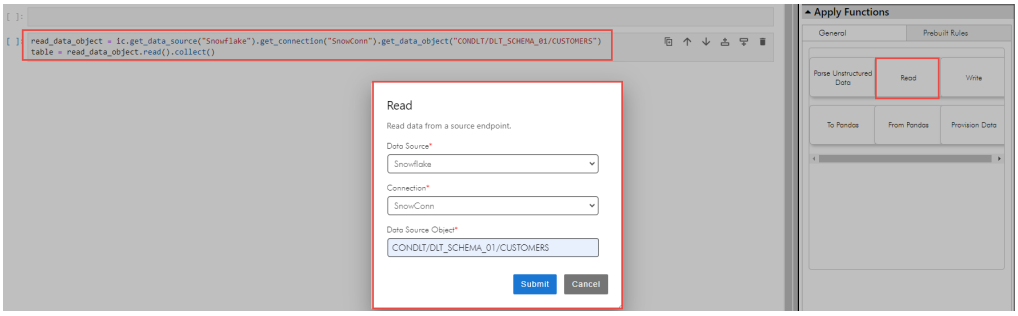
You can read from a data source and convert the data to a Pandas DataFrame.

You can configure the read and Pandas operations on the **Apply Functions** tab. When you configure a read operation, select the data source and connection, and then enter the data source object name from where you want to read data.

If you have already select the data source and connection, and then configure the read operation, the data source and connection are prepopulated in the Read operation dialog box.

In this example, we use a Snowflake connection to read customer data from a Snowflake source.

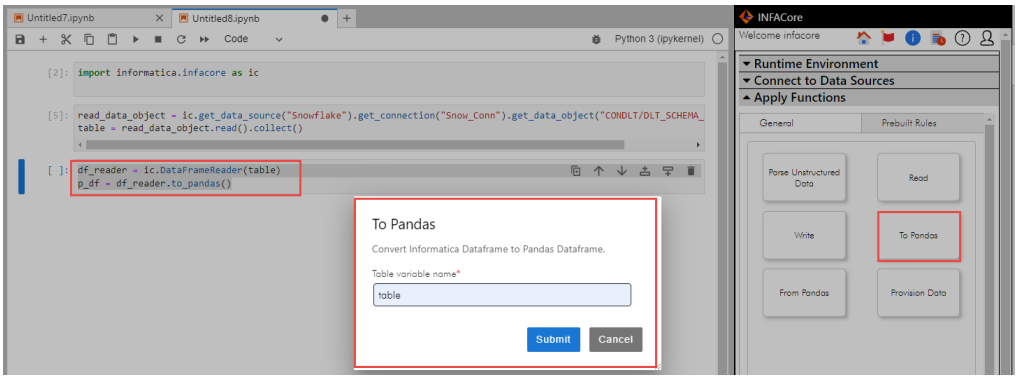
1. On the **General** tab, select **Read**.
2. Select the data source and the connection for the data source.
3. Enter the path to the source data object from where you want to read data.
4. Click **Submit**.



The following code appears in the JupyterLab workspace. You can run the code to read the data.

```
read_data_object =
ic.get_data_source("Snowflake").get_connection("SnowConn").get_data_object("CONDLT/
DLT_SCHEMA_01/CUSTOMERS")
table = read_data_object.read().collect()
```

5. To convert the data from the INFACore DataFrame to the Pandas DataFrame, select **To Pandas**.



6. Enter the variable name as table, and click **OK**.

The following code appears in the JupyterLab workspace. You can run the code to read the data.

```
df_reader = ic.DataFrameReader(table)
p_df = df_reader.to_pandas()
```

7. To print the results, enter the following code:

```
p_df.head()
```

When you run the code, the SDK returns the following data for the read operation:

	NAME	phone	email
0	John Doe	(408)-111-1234	john.doe@bluebird.dev
1	Jane Doe	(408)-111-1235	jane.doe@bluebird.dev
2	David Wright	(408)-222-1234	david.wright@dolphin.dev

To use the INFACore SDK for python directly to extract data from a data source, call the read() method on the DataObject instance and then invoke the to_pandas() method to convert the INFACore DataFrame to the Pandas DataFrame.

Convert to INFACore DataFrame and write data

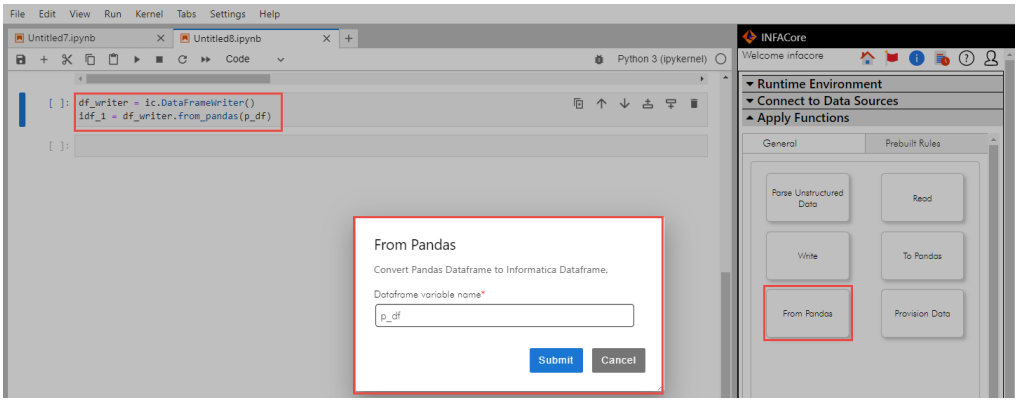
You can configure the Write operation to write data to a target endpoint. If the data is in the Pandas DataFrame, you can convert it to the INFACore DataFrame before you write to the target.

You can configure the write and the pandas operations on the **Apply Functions** tab.

You can choose to convert the data that you read from the Pandas DataFrame to the INFACore DataFrame, and then configure a write operation.

In this example, we use a Snowflake connection to write customer data to a Snowflake target.

1. To first convert the data from the Pandas DataFrame to the INFACore DataFrame, select **From Pandas**.
2. Enter the variable name as *p_df* for the Pandas DataFrame, and click **OK**.



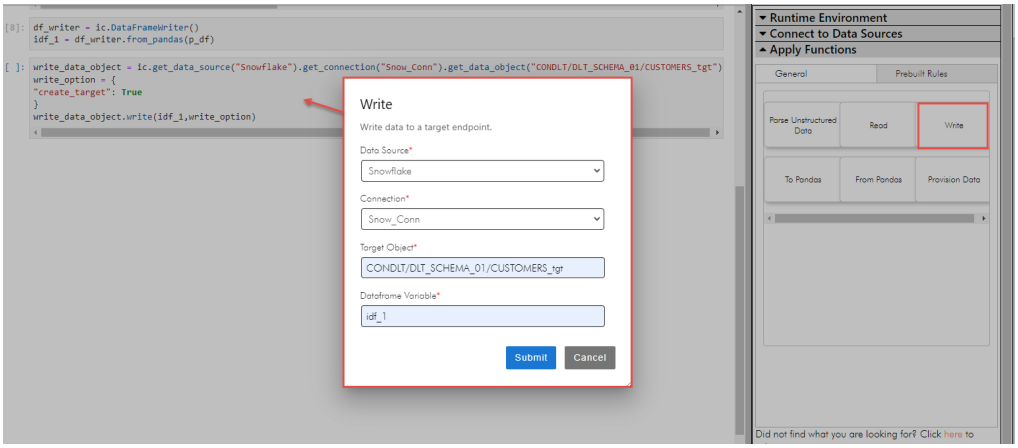
The following code appears in the JupyterLab workspace. You can run the code to convert the data from the Pandas DataFrame to the INFACore DataFrame.

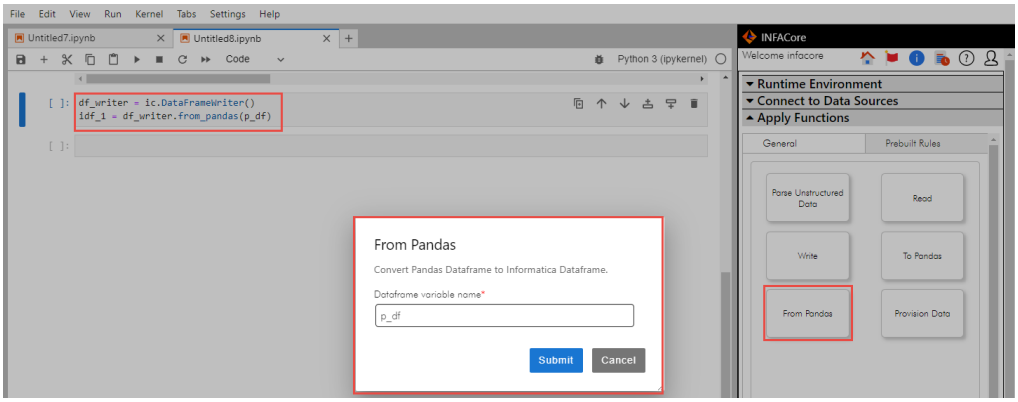
```
df_writer = ic.DataFrameWriter()
idf_1 = df_writer.from_pandas(p_df)
```

3. To write the data to the target, select **Write** on the **General** tab.
4. Select the data source and the connection to write to the data source.

Note: If you have already select the data source and connection, and then configure the write operation, the data source and connection are prepopulated in the Write operation dialog box.

5. Enter the path to the target data object to write the data.
6. Click **Submit**.





The following code appears in the JupyterLab workspace. You can run the code to write the data to the target.

```
write_data_object =
ic.get_data_source("Snowflake").get_connection("Snow_Conn").get_data_object("CONDLT/D
LT_SCHEMA_01/CUSTOMERS_tgt")
write_option = {
"create_target": True
}
write_data_object.write(idf_1,write_option)
```

When you run the code, the SDK writes the data to the target. The following code snippet shows sample target entries:

```
[{'@type': 'activityLogEntry',
  'id': '0154JUC100000000058X',
  'type': 'MTT',
  'objectId': '0154JU0Z0000000000BC',
  'objectName': 'mct_1676633702326',
  'runId': 1,
  'startTime': '2023-02-17T06:40:24.000Z',
  'endTime': '2023-02-17T06:41:58.000Z',
  'startTimeUtc': '2023-02-17T11:40:24.000Z',
  'endTimeUtc': '2023-02-17T11:41:58.000Z',
  'state': 1,
  'failedSourceRows': 0,
  'successSourceRows': 3,
```

```
'failedTargetRows': 0,
'successTargetRows': 3,
'startedBy': 'infacore_cai01',
'runContextType': 'REST_API_V2',
'entries': [{'@type': 'activityLogEntry',
  'id': '200105967',
  'type': 'MTT',
  'objectName': '',
  'runId': 1,
  'agentId': '0154JU0800000000000K',
  'runtimeEnvironmentId': '0154JU2500000000000K',
```

To use the INFACore SDK for python directly to write data to a data source, invoke the from_pandas() method to convert the Pandas DataFrame to the INFACore DataFrame,.and then call the write() method on the target DataObject instance.

Provision data

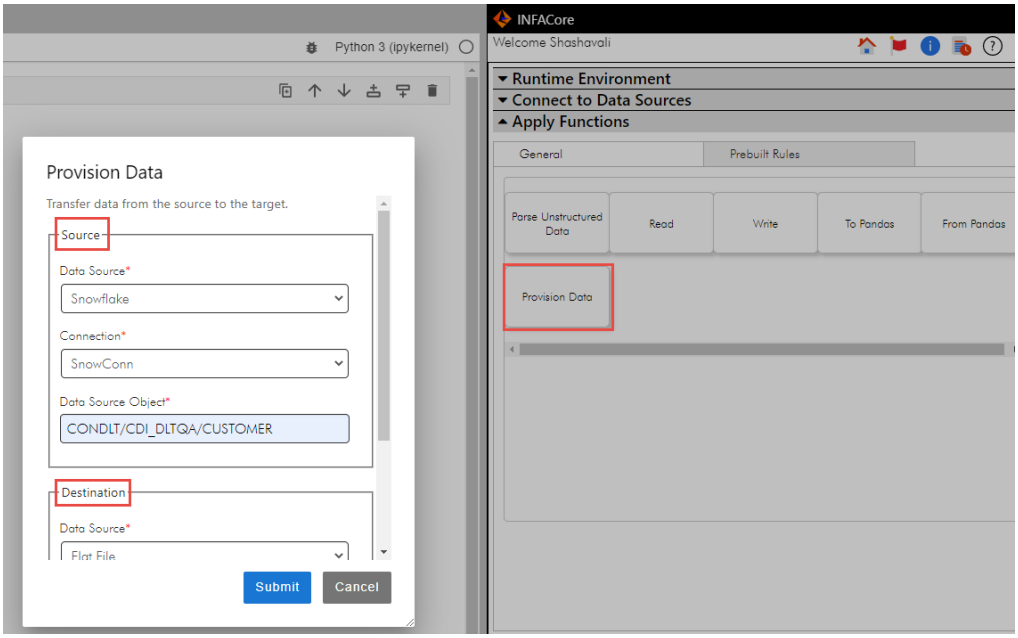
You can configure both the read and write operations together using the provision data function.

Select the configured source and target connections for the data source and destination. Provide the data object names in the source and destination. Run the Python code to read from and write data. The provision data function simplifies the effort of configuring separate read and write operations and saves time.

Configure the provision data operation on the **Apply Functions** tab.

In this example, we use a Snowflake connection to read customer data from a Snowflake source and write to a flat file target.

1. On the **General** tab, select **Provision Data**.
2. Select the data source name and connection, and enter the data source object for the source. Similarly, enter the details for the destination to write the data.



3. Click **Submit.**.

The following code appears in the JupyterLab workspace. You can run the code to read from and write the data.

```
read_data_object =
ic.get_data_source("Snowflake").get_connection("SnowConn").get_data_object("CONDLT/
CDI_DLTQA/CUSTOMER")
df = read_data_object.read()

write_data_object = ic.get_data_source("Flat
File").get_connection("FFile").get_data_object("FF_TG.csv")
write_option = {
"create_target": True
}
write_data_object.write(df,write_option)
```

Parse unstructured data

You can transform your input data into a user-defined structured format based on an intelligent structure model. You can use the **Parse Unstructured Data** function to analyze data such as log files, XML, or JSON files, Word tables, and other unstructured or semi-structured formats.

To parse unstructured data, use the **Parse Unstructured Data** function. The function uses Intelligent Structure Discovery to determine the underlying structure of the sample data file and creates a model of the structure.

Intelligent Structure Discovery creates the intelligent structure model based on a sample of your input data.

You can create models from the following input types:

- Text files, including delimited files such as CSV files and complex files that contain textual hierarchies
- Machine generated files such as weblogs and clickstreams
- JSON files
- XML files
- ORC files
- Avro files
- Parquet files
- Microsoft Excel files
- Data within PDF form fields
- Data within Microsoft Word tables
- XSD files

You can refine the intelligent structure model and customize the structure of the output data. You can edit the nodes in the model to combine, exclude, flatten, or collapse them.

You can process input from the source efficiently and seamlessly based on the intelligent structure model that you select. When you select the function, you associate it with the intelligent structure model. Select a data source based on a flat file to process local input files.

Configure the structure parser function

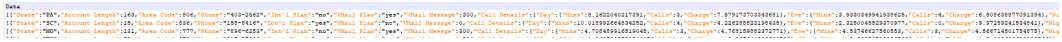
INFACore parses unstructured or semi-structured data using the Intelligent Structure Discovery (ISD) jars which is bundled with the INFACore installation.

To parse data, select the **Parse Unstructured Data** function, and specify the following fields:

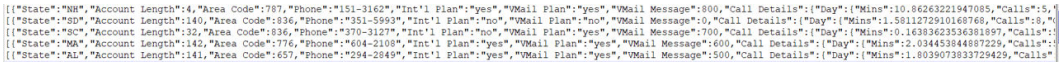
- **New DataFrame Name**: Specify a name for the new DataFrame. A DataFrame is a two-dimensional data structure, where data is aligned in a tabular fashion in rows and columns.
- **Schema file path**. Specify the file path to the sample schema file.
- **Input file path**. Specify the input file path of you source data that contains unstructured data.

Example

The following image is a snapshot of the unstructured data in JSON format in the `json_input.json` file that you want parse:



Provide the path to the sample schema `sample_schema.txt` file that you want INFACore to refer to parse the unstructured data:



See the sample Python code that displays when you apply the parser function with the input file and sample schema file:

```
import informatica.infacore as ic
pf = ic.ParserFunctions()
parser_data = pf.parse_unstructured_data("C:\\Users\\John\\Documents\\FF_SOURCES\
\json_input.json",
"C:\\Users\\John\\Documents\\FF_SOURCES\\sample_schema.txt")
```

To apply the Pandas function, invoke the Python SDK to convert the INFACore DataFrame to the Pandas DataFrame and return the rows:

```
df_reader = ic.DataFrameReader(parser_data)
p_df = df_reader.to_pandas()
p_df.head()
```

For more information, see the *INFACore SDK Reference for Python*.

When you run the code, the structure parser function returns data in a structured format:

State	Account Length	Area Code	Phone	Int'l Plan	VMail Plan	VMail Message	token	Mins	Calls	Charge	CustServ Calls	Churn
PA	163	806	403-2562	no	yes	300	Day	8.162204	3	7.579174	3	True.
PA	163	806	403-2562	no	yes	300	Eve	3.933035	4	6.508639	3	True.
PA	163	806	403-2562	no	yes	300	Night	4.065759	100	5.111624	3	True.
PA	163	806	403-2562	no	yes	300	Intl	4.92816	6	5.673203	3	True.
SC	15	836	158-8416	yes	no	0	Day	10.018993	4	4.226289	8	False.

Apply pre-built rules

You can apply pre-built rules to evaluate the accuracy and validity of your data.

For example, you can determine the accuracy of the input addresses, fix errors in the addresses, and enhance the addresses where possible with additional information. You can also measure and report on the quality of each address, retrieve country-specific enrichments for addresses, and verify addresses to the certification standards that a country defines.

The in-built rules evaluate and update input data and return the results.

Select the required prebuilt function from the **Apply Functions** section in INFACore. You need to specify the data object variable and the required column name for which you want to apply the function. For information about how to create the data object variable, see #unique_23/unique_23_Connect_42_GUID-23C83356-8E09-4ECA-A67A-CF00C885784Don page 21.

Get country ISO values

You can apply this function to your data to standardize country names to their respective country codes.

To perform this operation, select the **Get Country ISO Values** function, and specify the data object variable. Then, enter the name of the country column for which you want the ISO values.

The function reads the input country name and returns the full ISO country name and their respective ISO two-letter or three-letter country codes.

In the following example, the input flat file includes a column named *country* and contains the following data: *India*, *XYZ*, and *123!*.

The following example is the input code snippet when you apply the function::

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.get_country_iso_values(FF_DV,"country")
```

The following snippet shows the output:

```
Out_Country_Name: string
Out_Country_ISO2Char: string
Out_Country_ISO3Char: string
----
Out_Country_Name: [["INDIA","XYZ","123!"]]
Out_Country_ISO2Char: [["IN","XYZ","123!"]]
Out_Country_ISO3Char: [["IND","XYZ","123!"]]
```

Convert diacritic English characters

You can apply this function to translate diacritic characters to regular ASCII text characters.

To perform this operation, select the **Convert Diacritic English Chars** function, and specify the data object variable. Then, enter the column name that contains the diacritic characters.

The function changes the diacritic characters to regular ASCII text. For example, when you apply this function to a column named tëst, the function changes the ë to an e and returns tëst as test.

In the following example, the input flat file includes a column named *diacritic_chars* and contains diacritic English characters:

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.convert_diacritic_english_chars(FF_DV,"diacritic_chars")
```

The following snippet shows the output, where the diacritic characters are replaced with ASCII text characters:

```
In_Field: string
Out_Field: string
----
In_Field: [["tëst","ÔN","\uXXXX"]]
Out_Field: [["test","ON","\uXXXX"]]
```

Remove control characters

You can apply this function to identify and remove non-printable characters in the source data.

To perform this operation, select the **Remove Control Characters** function, and specify the data object variable. Then, enter the column name that contains the control characters.

The function removes the control characters. For example, the input flat file includes a column named *characters* that contains control characters.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.remove_control_characters(FF_DV,"Characters")
```

The following snippet shows the output, where the control characters are removed:

```
Input_Field: string
Output_Field: string
----
Input_Field: [["    "," ","→"]]
Output_Field: [["","  ","→"]]
```

Parse name

You can parse the first name and surname and the respective gender to determine the gender score and gender status for the person name.

If you know the gender for a name, then the function uses the gender specific score to determine the gender. Acceptable input for male and female genders is M and F.

If you do not know the gender, the function uses the highest of the male or female scores to determine the status. The rule also calculates the probable gender based on the first name input and provides a confidence score based on the frequency a name occurs as male or female.

Genders are only assigned a score if the probability of the gender being male or female is 70% or more. Unknown genders always have a confidence score of zero.

To perform this operation, select the **Parse Name** function, and specify the data object variable. Then, enter the first name, surname, and gender column names that you want to parse.

For example, the input flat file includes columns for first names, surnames, and gender.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
result=dqf.parse_name(FF_DV,"FirstName","Surname","Gender")
df_reader = ic.DataFrameReader(result)
p_df = df_reader.to_pandas()
p_df.head()
```

The function parses the data and returns the following gender score and gender status:

```
In_Firstname: [["James","Mary","Ishika"]]
In_Surname: [["Thomson","Patricia","Garg"]]
In_Gender: [["M","F","F"]]
R1_male_first_name_prob: [[0.9611602416284911,0.862213129336417,0.5]]
R1_female_first_name_prob: [[0.09618367642795013,0.9994692271687158,0.9994110718492345]]
R2_male_surname_prob: [[0.9810939357907253,0.06919354838709678,0.9996677740863787]]
R2_female_surname_prob: [[0.9999596024884867,0.0002336138151259772 2,0.9996677740863787]]
R_male_name_parse_prob: [[0.9999983675038314,0.31748599893598634,0.9996677740863787]]
R_female_name_parse_prob: [[0.999620522413733,0.9929068110415156,0.9999998041624873]]
Status: [["Probably Valid","Uncommon Name","Probably Valid"]]
```

Standardize the United States company names

You can apply rules to standardize United States company names and provide acronyms for the acronyms if possible.

To perform this operation, select the **Standardize US Companyname** function, and specify the data object variable. Then, enter the column name that contains the United States company names that you want to standardize.

For example, the input flat file includes a column named *company* that contains few company names in the Unites States.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.standardize_us_companyname(FF_DV,"company"))
```

The following snippet shows the output with the standardized company names:

```
Input_Company_Name: string
Out_CompanyName_Std: string
Out_CompanyName_Acronym: string
----
Input_Company_Name: [["Informatica","ABC","Amazon"]]
Out_CompanyName_Std: [["Informatica Corp.","American Broadcasting Co.","Amazon"]]
Out_CompanyName_Acronym: [["Informatica Corp.","American Broadcasting Co.","Amazon"]]
```

Validate names

You can apply this function to flag suspicious or fake names in the source data.

To perform this operation, select the **Find Invalid Names** function, and specify the data object variable. Then, enter the column name that contains names that you want to validate.

When you run the code, the function validates the names.

For example, the input flat file includes a *Name* column that contains control characters.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.find_invalid_names(FF_DV,"Name")
```

The following snippet shows the output, where the function flags the names that are valid and those that are suspected:

```
Name: string
Validation_Status: string
----
Name: [["Arcane","Aquata","ARTHUR M LYNCH JR"]]
Validation_Status: [["Valid","Suspect","Valid"]]
```

Validate email address

You can apply this function to check if the email format in the data source is valid. The function does not check if the email is an active address.

To perform this operation, specify the data object variable, and then enter the column name that contains the email address that you want to validate.

When you run the code to validate the email format, the function returns values as valid or not valid.

For example, the input flat file includes a column named *email* that contains the email addresses that you want to validate.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.validate_email_address(FF_DV,"email")
```

The following snippet shows the output returned with the validation values for the email address:

```
Email_Id: string
Validation_Status: string
----
Email_Id: [["foobar@informatica.com","123","foobar"]]
Validation_Status: [["Valid","Invalid","Invalid"]]
```

Validate SSN

You can apply this function to parse a United States Social Security Number (SSN) pattern from a large string of text.

The function parses SSNs with dashes or without dashes, formats, and validates the SSN. Format options include without punctuation, with punctuation, and spaces.

To perform this operation, specify the data object variable, and then enter the column name that contains the SSNs that you want to validate.

By default, the rule writes Social Security Numbers without any punctuation.
The rule can output the following formats:

- No Punctuation - nnnnnnnnn
- Space - nnn nnn nnn
- Dash - nnn-nnn-nnn

To change the standardization format, open the **dq_SSN_Format** transformation in the rule and update the expression on the **SSN_Format** port.

When you run the code to validate the SSN, the function returns values as valid or not valid.

For example, the input flat file includes a column named *SSN* that contains the United States SSN.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.validate_ssn(FF_DV,"ssn")
```

The following snippet shows the output returned with the validation values for the SSN:

```
Input_SSN: string
Out_SSN: string
Out_SSN_Status: string
Out_SSN_Status_Message: string
Out_SSN_Score: double
Out_Remaining_Text: string
----
Input_SSN: [["532459641","680-11-2943","41735abc55555"]]
Out_SSN: [["532459641","680112943","41735abc5"]]
Out_SSN_Status: [["Valid","Valid","Invalid"]]
Out_SSN_Status_Message: [["Valid structure, valid group and area values.","Valid
structure,
valid group and area values.","Invalid SSN length"]]
Out_SSN_Score: [[0.9,0.9,0.25]]
Out_Remaining_Text: [["","","41735abc55555"]]
```

Validate the Unites States zip code

You can validate if the input data is a five-digit United States zip code.

To perform this operation, select the **Validate USA Zipcode** function, and specify the data object variable. Then, enter the column name that contains the United States zip codes that you want to validate.

When you run the code, the function validates if the zip code is valid.

For example, the input flat file includes a column named *zipcode* that contains the Unites States zip codes.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.validate_usa_zipcode(FF_DV,"zipcode")
```

The following snippet shows the validation values for the zip codes:

```
ZIPCode: string
Validation_Status: string
----
ZIPCode: [["99501","000","12340"]]
Validation_Status: [["Valid","Invalid","Invalid"]]
```

Validate a state in the United States

You can apply this function to verify if the entry is a valid state in the United States.

To perform this operation, select the **Validate USA State** function, and specify the data object variable. Then, enter the column name that contains the state names that you want to validate.

When you run the code, the function validates the state name.

For example, the input flat file includes a column named *state* that contains the names of the states in the Unites States.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.validate_usa_state(FF_DV,"state")
```

The following snippet shows the output returned with the validation values for the states:

```
State: string
Out_State_Status: string
----
State: [["Punjab","California","California1"]]
Out_State_Status: [["Invalid","Valid","Invalid"]]
```

Validate the United States counties

You can apply this function to validate if the input string is a valid county name in the United States.

To perform this operation, select the **Validate US County** function, and specify the data object variable. Then, enter the column name that contains the county names that you want to validate.

When you run the code, the function validates if the county name is valid.

For example, the input flat file includes a column named *county* that contains the names of the Unites States counties.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.validate_us_county(FF_DV,"County")
```

The following snippet shows the output returned with the validation values for the counties:

```
County: string
County_Status: string
----
County: [["Jackson County","Marion","Clay"]]
County_Status: [["Invalid","Valid","Valid"]]
```

Validate the United States phone number

You can apply this function to validate and enrich telephone numbers in the United States.

To perform this operation, select the **Validate USA Phone Number** function, and specify the data object variable. Then, enter the column name that contains the phone numbers that you want to validate.

When you run the code, the function validates if the phone number is a valid United States number and also enriches the phone number.

In the following example, the input flat file includes a column named *phnNumber* that contains the names of the Unites States phone numbers.

The following snippet is the input code when you apply the function:

```
import informatica.infacore as ic
FF_DV = ic.get_data_source("Flat
File").get_connection("DR_FlatFile").get_data_object("input.csv")
dqf = ic.DataQualityFunctions()
dqf.validate_usa_phone_number(FF_DV,"phnNumber")
```

The following snippet shows the output returned with the validation values for the phone number:

```
Input_Phone: string
Out_Phone_w_Extension_Std: string
Out_Phone_Std: string
Out_Phone_Dashes: string
Out_Phone_No_Spaces: string
Out_Phone_Extension: string
Out_Phone_Status: string
Out_Phone_Status_Msg: string
Out_Phone_Quality_Score: double
Out_Remaining_Text: string
----
Input_Phone: [["(555) 555-1234","+15853042461","+911234567890"]]
Out_Phone_w_Extension_Std: [["","(585) 304-2461",""]]
Out_Phone_Std: [["","(585) 304-2461",""]]
Out_Phone_Dashes: [["","585-304-2461",""]]
Out_Phone_No_Spaces: [["","5853042461",""]]
Out_Phone_Extension: [["","",""]]
Out_Phone_Status: [["Invalid","Valid","Invalid"]]
Out_Phone_Status_Msg: [["Area code invalid.","no error","Area code invalid."]]
Out_Phone_Quality_Score: [[40,95,40]]
Out_Remaining_Text: [["(","+1","+91"]]
```

CHAPTER 5

Monitoring your activities and logs

You can monitor activities that you and other people have performed in your organization. You can get the status of your operations and view the logs to troubleshoot any issues you encounter.

You can monitor your activities and status of the INFACore operations on the following pages:

Activity Log

The **Activity Log** page lists all your activities performed in INFACore, along with the timestamp. Use this page for live monitoring of your activities performed in your organization.

Download Logs

The **Download Logs** page lists all the Python SDK logs and the INFACore extension logs. It also logs the methods invoked, REST call requests and responses, errors, and exceptions. Use this page for failure analysis and debugging of the jobs in your organization.

INDEX