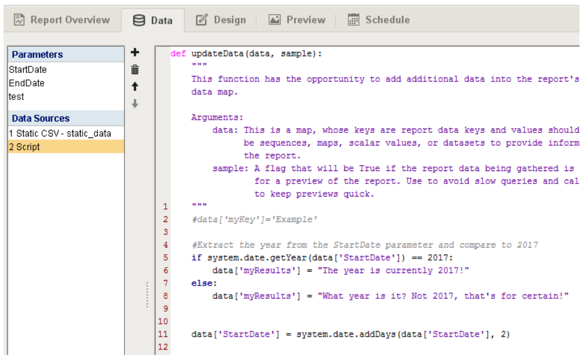
Scripting Data Source

Script Data Source

The Script data source allows you to use scripting to add additional data into a report, or modify existing data. With this data source, you to can pull in data from the database and then modify it using a script before pushing it out as a data key for use in the report.



```
def updateData(data, sample):
    """
    This function has the opportunity to add additional data into the report's
    data map.

    Arguments:
        data: This is a map, whose keys are report data keys and values should
              be sequences, maps, scalar values, or datasets to provide inform
              the report.
        sample: A flag that will be True if the report data being gathered is
                for a preview of the report. Use to avoid slow queries and cal
                to keep previews quick.
    """
    #data['myKey']='Example'

    #Extract the year from the StartDate parameter and compare to 2017
    if system.date.getYear(data['StartDate']) == 2017:
        data['myResults'] = "The year is currently 2017!"
    else:
        data['myResults'] = "What year is it? Not 2017, that's for certain!"


    data['StartDate'] = system.date.addDays(data['StartDate'], 2)
```

On this page

...

INDUCTIVE UNIVERSITY

Data Sources - Scripting

Watch the Video

Accessing Parameters and Data Sources

One of the main uses of the Scripting Data Source is to allow for data access and manipulation as the report is being generated, allowing you to replace the original results, or add new additional content.

The synatx for creating or referencing a data key in the Script datasource is described below:

Pseudocode - Data Key Syntax

```
data['keyName']
```

Order Matters

As mentioned on the Report Data page, the order of Data Sources determines which parameters and data sources they may reference.

Because of this, it is **highly recommended** that Scripting Data Sources are placed at the bottom of the Data Sources list.

Creating a New Key

To create a new key that the report can use:

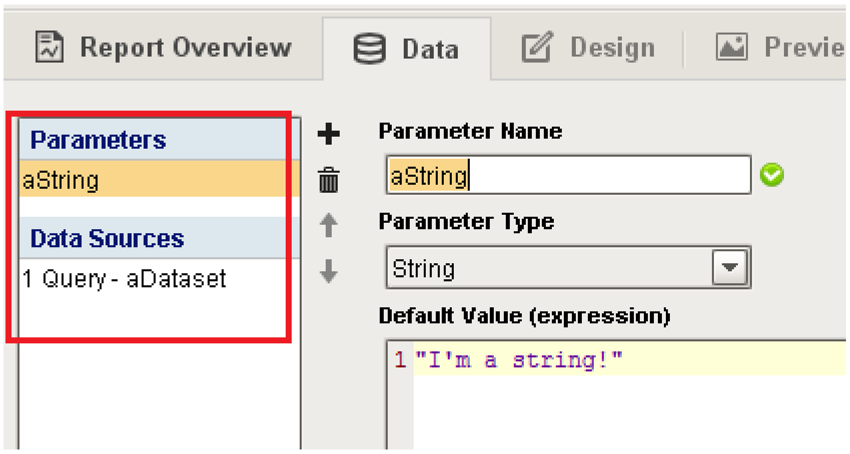
Python - Creating a New Key

```
data['newKey'] = "This key was generated on the fly!"
```

The type of data assigned to the key determines where it appears in the Design Panel.

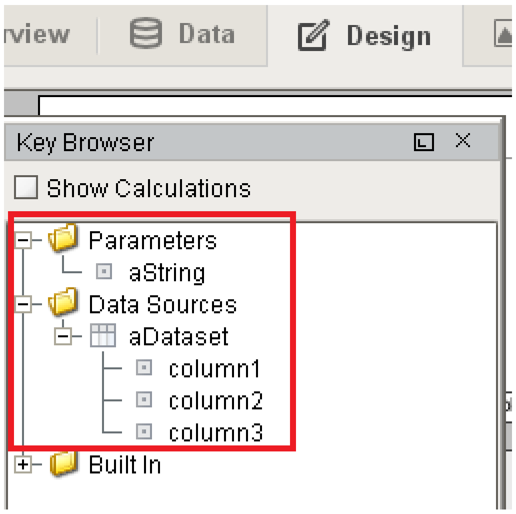
- Simple data types, like strings and integers, will appear as **Parameters.**
- Datasets will appear as **Data Sources.**

Report Data



```
"I'm a string!"
```

Key Browser



Parameters

Where the 'keyName' is the name of your data key. Thus, reading the value of a parameter, such as the initial StartDate parameter, can be accomplished by using system.date.getYear() and an if-statement.

Python - Accessing a Parameter's Value

```
# Extract the year from the StartDate parameter and compare to 2017.
if system.date.getYear(data['StartDate']) == 2017:
        # Do work here if the year lines up.
```

Of course, we can write back to the key and override it's value:

Python - Overriding the Default StartDate Parameter

```
# This line would override the default StartDate parameter by add adding a day.
data['StartDate'] = system.date.addDays(data['StartDate'], 1)
```

Additionally, this allows you to expand the Accessing a Parameter's Value example above by creating a new key:

Python - Accessing a Parameter's Value

```
# Extract the year from the StartDate parameter and compare to 2017.
if system.date.getYear(data['StartDate']) == 2017:
        # Create a new key and assign it one value if our condition is true...
        data['myResults'] = "The year is currently 2017!"
else:
        # ...or assign a different value if the condition is false. This way we can always assume the
key 'myResults' exists.
        data['myResults'] = "What year is it? Not 2017, that's for certain!"
```

Data Sources

Static CSVs

While uncommon, Static CSVs may be accessed in a Scripting Data Source. The syntax is similar to working with Parameters.

Python - Static CSV example

```
# Take a Static CSV data source, and replicate its contents in a new key.
data['static_data'] = data['Area Data']
```

Query-Based Data Sources

Reading the contents of a query-based Data Source, such as a SQL Query Data Source or Tag Historian Query, requires the `getCoreResults()` function, which returns the results in a standard dataset:

Python - Accessing a Query-Based Data Source's Value

```
# Query-based Data Sources are slightly different than parameters, so we must use getCoreResults() to
extract the data.
rawResults = data['keyName'].getCoreResults()

# getCoreResults() returns a dataset, so we can utilize getValueAt() and rowCount within our scripts.
resultsCount = data['keyName'].getCoreResults().rowCount
```

When working with data sources, it is unusual to attempt to write back, since datasets are immutable. Instead, the prefered approach is to create a new key with the modified results.

Say we have a query data source named "**Area Data**" which contains four columns: **month**, **north_area**, **south_area**, and **t_stamp**. If we need to build a new data source without the t_stamp column, we can use the following code:

Python - Building a New Data Source

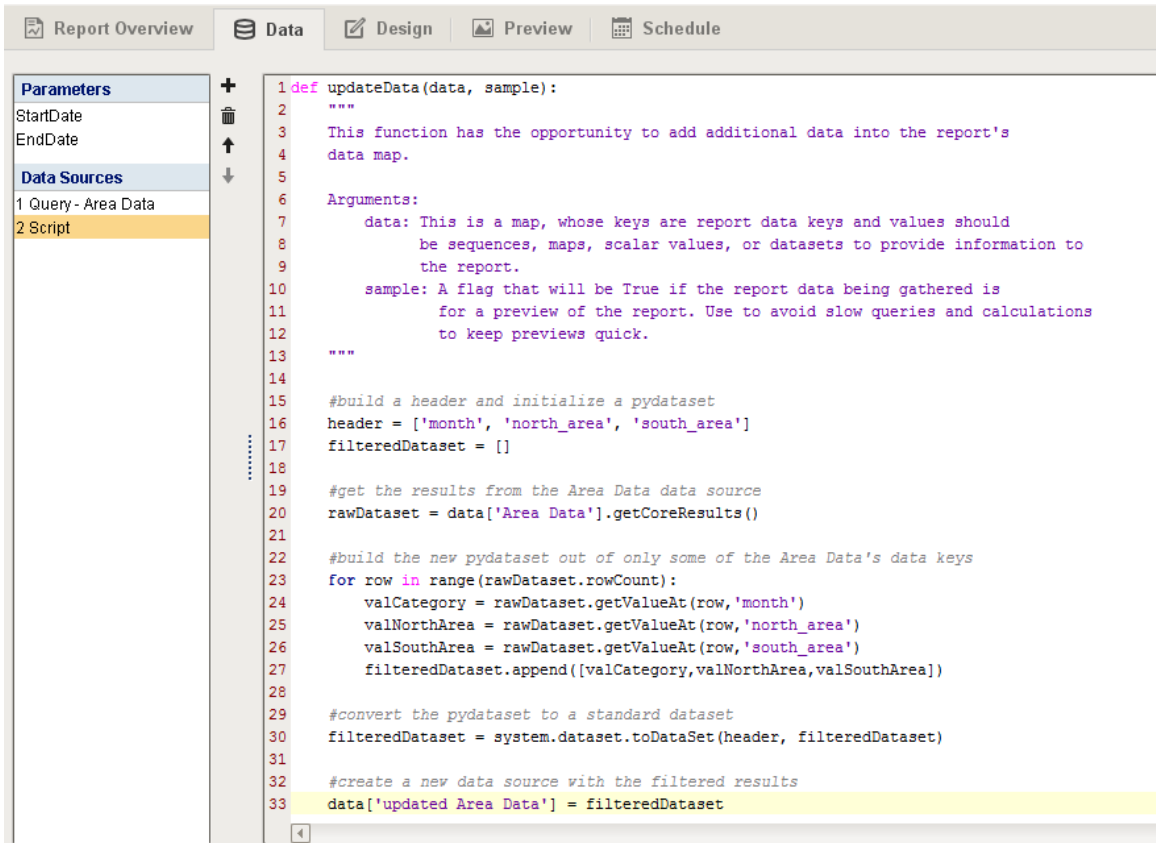
```
# build a header and initialize a pydataset
header = ['month', 'north_area', 'south_area']
filteredDataset = []

# get the results from the Area Data data source
rawDataset = data['Area Data'].getCoreResults()

# build the new pydataset out of only some of the Area Data's data keys
for row in range(rawDataset.rowCount):
        valCategory = rawDataset.getValueAt(row,'month')
        valNorthArea = rawDataset.getValueAt(row,'north_area')
        valSouthArea = rawDataset.getValueAt(row,'south_area')
        filteredDataset.append([valCategory,valNorthArea,valSouthArea])

# convert the pydataset to a standard dataset
filteredDataset = system.dataset.toDataSet(header, filteredDataset)

# create a new data source with the filtered results
data['updated Area Data'] = filteredDataset
```



```
def updateData(data, sample):
    """
    This function has the opportunity to add additional data into the report's
    data map.

    Arguments:
        data: This is a map, whose keys are report data keys and values should
              be sequences, maps, scalar values, or datasets to provide information to
              the report.
        sample: A flag that will be True if the report data being gathered is
                for a preview of the report. Use to avoid slow queries and calculations
                to keep previews quick.
    """

    #build a header and initialize a pydataset
    header = ['month', 'north_area', 'south_area']
    filteredDataset = []

    #get the results from the Area Data data source
    rawDataset = data['Area Data'].getCoreResults()

    #build the new pydataset out of only some of the Area Data's data keys
    for row in range(rawDataset.rowCount):
        valCategory = rawDataset.getValueAt(row,'month')
        valNorthArea = rawDataset.getValueAt(row,'north_area')
        valSouthArea = rawDataset.getValueAt(row,'south_area')
        filteredDataset.append([valCategory,valNorthArea,valSouthArea])

    #convert the pydataset to a standard dataset
    filteredDataset = system.dataset.toDataSet(header, filteredDataset)

    #create a new data source with the filtered results
    data['updated Area Data'] = filteredDataset
```

Nested Queries

What if our '**Area Data**' query has a nested query called '**Area Details**' that we would like to manipulate in a script? This is useful when using Table Groups.

Python - Script data source for nested query

```
nested = data['Area Data'].getNestedQueryResults()    # Gets results from our parent query
subQuery = nested['Area Details']    # Gets the subquery we want -- there can be more than one
header = ['productName', 'cost', 'triple']
alteredDataset = []
for child in subQuery:
        children = child.getCoreResults()    # children is a dataset
        for row in range(children.rowCount):
                valProductName = children.getValueAt(row,'productName')
                valCost = children.getValueAt(row,'cost')
                valTimesThree =  None
                if valCost != None:
                        valTimesThree = 3 * valCost
                alteredDataset.append([valProductName,valCost,valTimesThree])

# convert the pydataset to a standard dataset
alteredDataset = system.dataset.toDataSet(header, alteredDataset)

# create a new data source with the altered results
data['Updated Area Details'] = alteredDataset
```