



GraphQL Quick Start Guide

SL1 version 8.12.1

Overview

This manual describes the functionality of the SL1 GraphQL API and is intended for developers who are responsible for integrating SL1 with external systems. To use this manual, you should have a general understanding of the HTTP protocol.

This manual includes the following topics:

<i>What is the SL1 GraphQL API?</i>	3
<i>Overview of the GraphQL User Interface</i>	3
<i>Example: Querying for a List of Devices</i>	5
<i>Querying the ScienceLogic GraphQL API from an External Application</i>	15

What is the SL1 GraphQL API?

The **SL1 GraphQL API** allows external systems to programmatically access data in SL1 using the GraphQL query language. The API gives access to entities in the platform, such as devices and collected data, using standard HTTP request/response protocols. For more information about GraphQL, see the [GraphQL documentation](#).

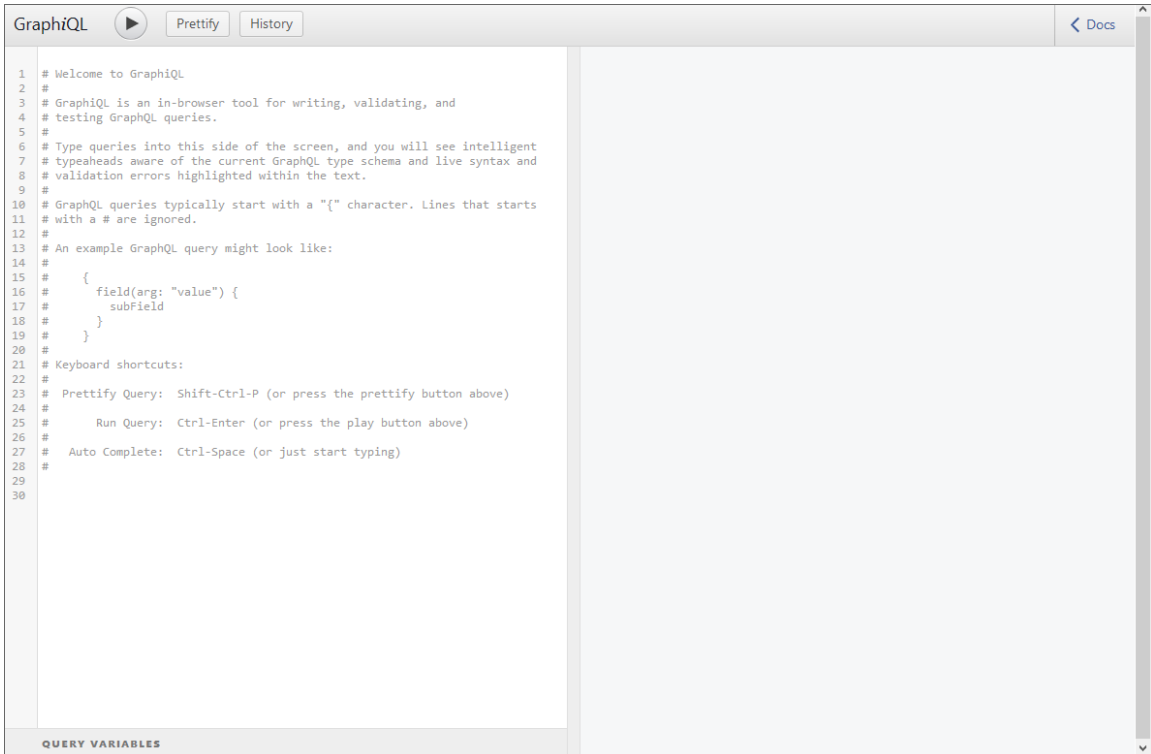
Overview of the GraphiQL User Interface

GraphiQL is a user interface for interactively exploring the capabilities of, and executing queries against, a GraphQL API. Unlike the SL1 REST API, all requests to the GraphQL API on a given SL1 system use the same URI.

NOTE: GraphiQL is not maintained by ScienceLogic, and you can access its documentation at <https://github.com/graphql/graphiql>.

To create a sample GraphQL query:

1. To access the GraphiQL interface, type the URL or IP address for SL1 in a browser, add **/gql** to the end of the URL or IP address, and press **[Enter]**. The GraphiQL interface appears:



The screenshot shows the GraphiQL web interface. At the top, there are buttons for 'Prettify' and 'History', and a 'Docs' link on the right. The main area contains a text editor with the following content:

```
1 # Welcome to GraphiQL
2 #
3 # GraphiQL is an in-browser tool for writing, validating, and
4 # testing GraphQL queries.
5 #
6 # Type queries into this side of the screen, and you will see intelligent
7 # typeheads aware of the current GraphQL type schema and live syntax and
8 # validation errors highlighted within the text.
9 #
10 # GraphQL queries typically start with a "{" character. Lines that starts
11 # with a # are ignored.
12 #
13 # An example GraphQL query might look like:
14 #
15 #   {
16 #     field(arg: "value") {
17 #       subField
18 #     }
19 #   }
20 #
21 # Keyboard shortcuts:
22 #
23 #   Prettify Query: Shift-Ctrl-P (or press the prettify button above)
24 #
25 #   Run Query: Ctrl-Enter (or press the play button above)
26 #
27 #   Auto Complete: Ctrl-Space (or just start typing)
28 #
29 #
30 #
```

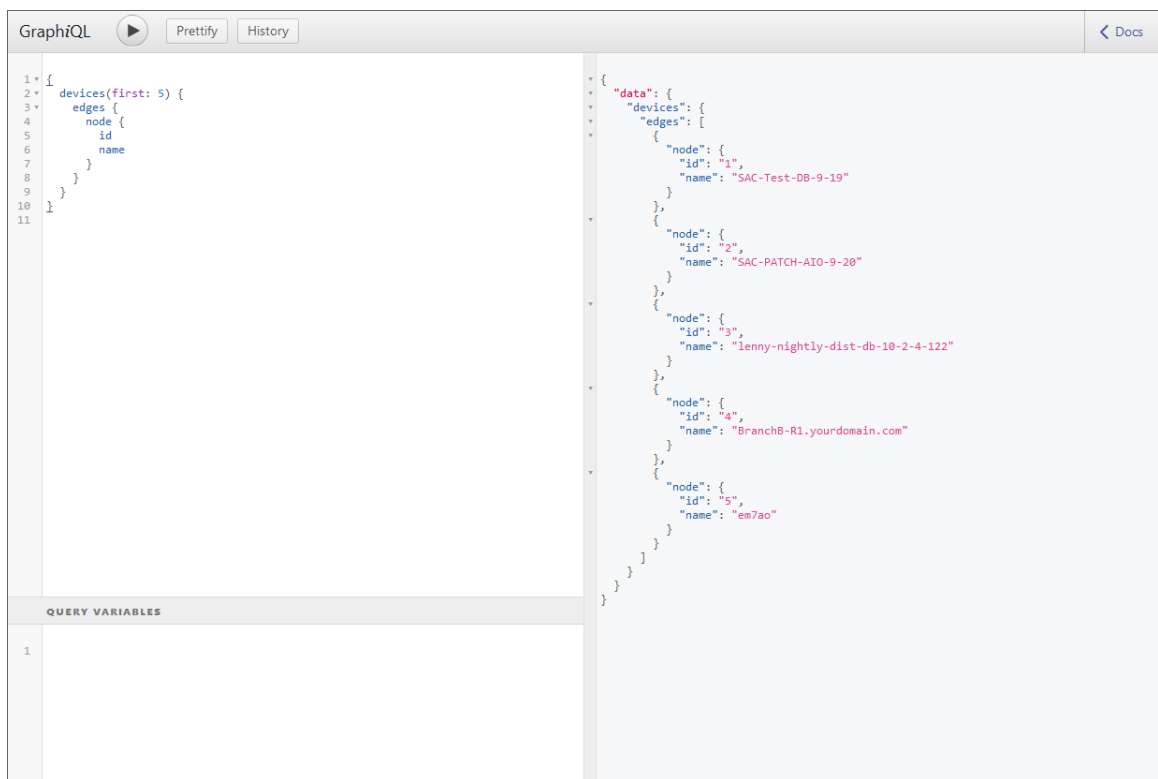
At the bottom of the interface, there is a section labeled 'QUERY VARIABLES'.

2. In the main query pane, type the following query to return the ID and name of the first five devices in the SL1 system:

```
query { devices (first:5) { edges { node { id name } } } }
```

TIP: Click the **[Prettify]** button to format the query and add syntax highlighting to make the query easier to read. Note that the *Prettify* process removes the `query` syntax if only one query is present in the main query pane.

3. Click the **[Execute Query]** (Play) button. The query executes, and the results of the query appear in the right pane:



The GraphQL user interface includes the following elements:

- The main query pane is a text area for entering GraphQL queries.
- The **[Execute Query]** (Play) button executes the query.
- The **[Prettify]** button re-formats the text in the main query pane.
- The **[History]** button opens a new pane that contains a list of the previous queries run in the GraphQL interface. Click a query to add it to the main query pane.
- The lower-left pane is a text area for entering variables that can be used when executing queries.
- The right pane displays the result of the last executed query.

- The **[Docs]** button expands the **Documentation Explorer** pane. The **Documentation Explorer** pane displays information about how to query the SL1 GraphQL API.

Example: Querying for a List of Devices

The query you executed in the previous section returns the ID and name of the first five devices:

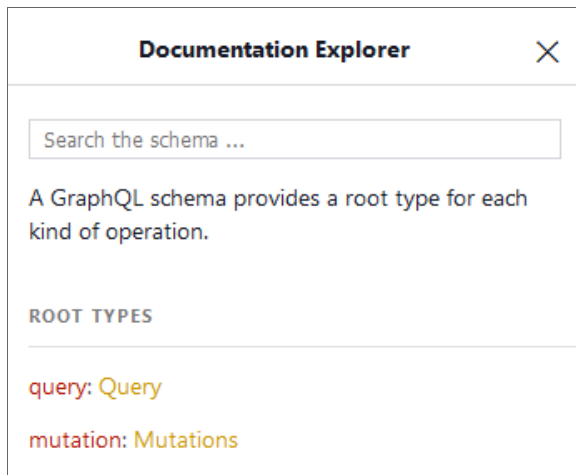
```
query {
  devices (first:5) {
    edges {
      node {
        id
        name
      }
    }
  }
}
```

This section describes how to use the **Documentation Explorer** pane to:

- Construct this query.
- Add an additional field to this query.
- Add additional parameters to the query so that you can iteratively request all devices in the SL1 system.

Perform the following steps:

1. If you have not already expanded the **Documentation Explorer** pane, click the **[Docs]** button. The two root types of the GraphQL API are displayed:



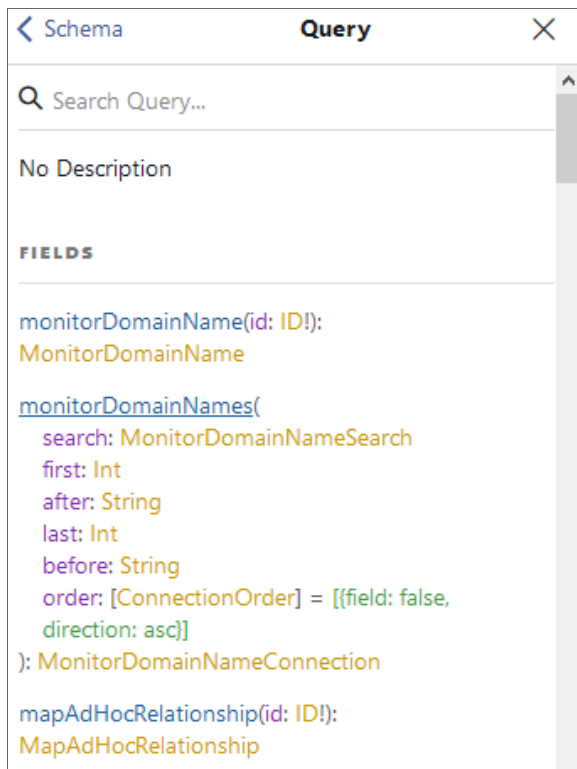
The root types describe the types of operations that can be performed using the SL1 GraphQL API:

- **Query**. A query operation specifies a set of data to *retrieve* from SL1 . If the query is successful, the API returns that set of data.
- **Mutation**. A mutation operation specifies a *change* to an element in SL1 . If the mutation is successful, an element in SL1 is created, edited, or deleted.

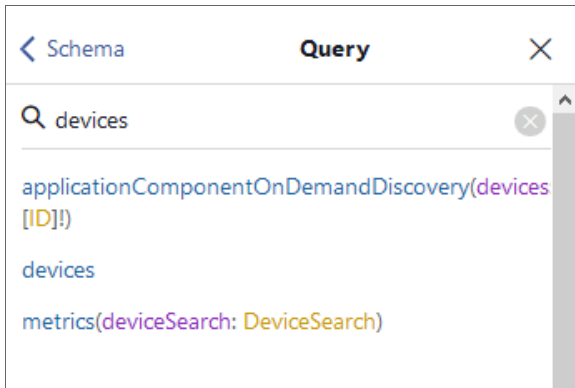
The example query specifies a query operation. The root type definition is highlighted in bold:

```
query {  
  devices (first:5) {  
    edges {  
      node {  
        id  
        name  
      }  
    }  
  }  
}
```

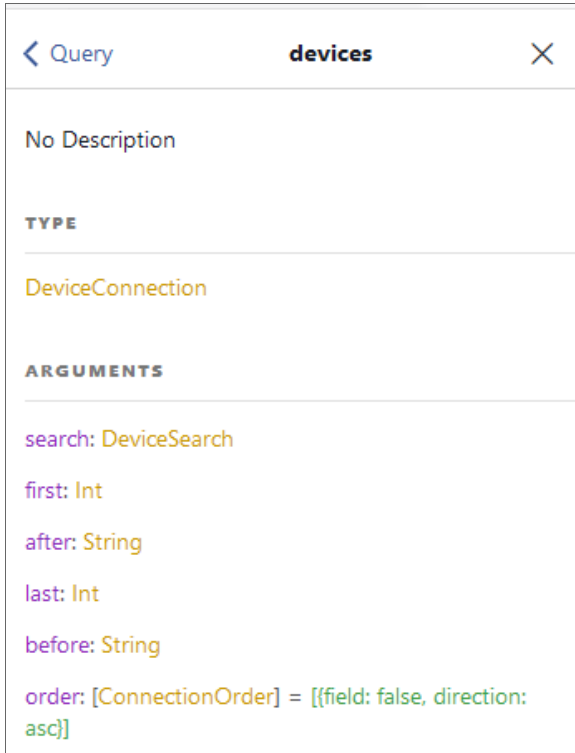
2. In the **Documentation Explorer** pane, click **Query**. A list of query definitions, which specify how you can query for different types of data, appears:



3. In the **Search Query** field, type "devices":



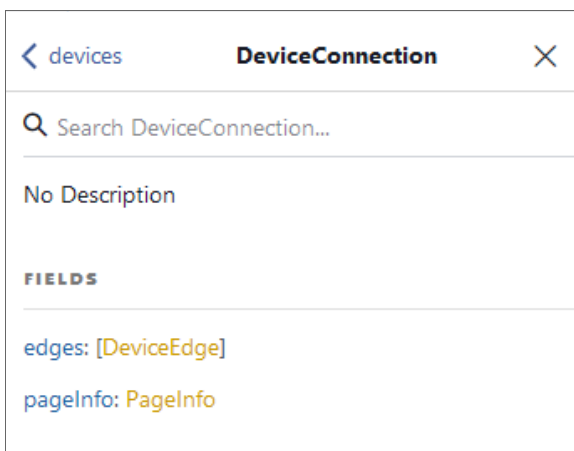
4. Select **devices** from the search results. The "devices" query definition appears:



The list of parameters in parentheses specify how you can sort and filter the data in the response. The example query uses the "devices" query definition with the "first" parameter (an integer) to limit the number of devices included in the response to five:

```
query {
  devices (first:5) {
    edges {
      node {
        id
        name
      }
    }
  }
}
```

5. Click *DeviceConnection* to view the data you can request using this query definition:

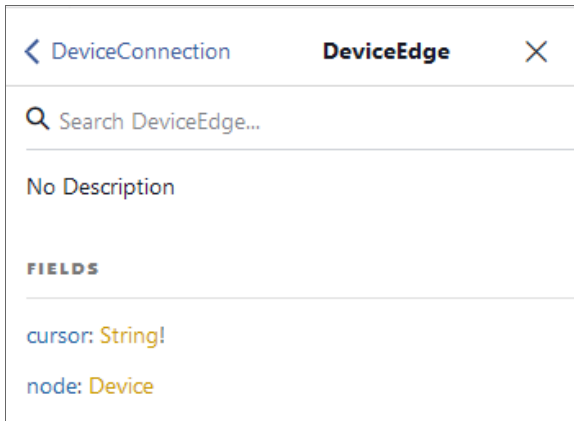


The *DeviceConnection* object has two fields, both of which are also objects. The "edges" field is the object that defines the structure of the entity data (in this example, the structure of device attributes). The "pageInfo" field is the object that defines metadata about the returned list of values, such as the first and last values returned in the list of values in the response.

The example query does not request any metadata, so only the edges field is included in the query:

```
query {
  devices (first:5) {
    edges {
      node {
        id
        name
      }
    }
  }
}
```

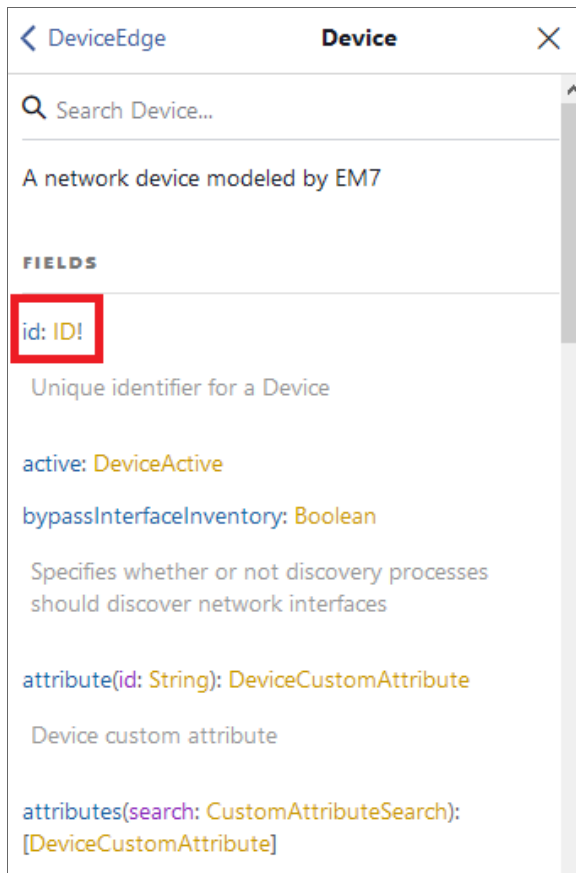

6. Click "DeviceEdge" to view how to structure the edges section of the query:



The *DeviceEdge* object has two fields. For objects that will be included in a list in a response, the "cursor" field is a string that specifies the position of the entry in the list, and the "node" field is the object that specifies the collection of data fields and objects associated with each list entry. The example query does not request any metadata, so only the node field is included in the query:

```
query {
  devices (first:5) {
    edges {
      node {
        id
        name
      }
    }
  }
}
```

7. Click "Device" to view how to structure the "node" section of the query:



The example query includes the "id" fields from this list. If you select the type information for the ID field, the description specifies that this field is a scalar type. **Scalar types** do not have subordinate fields:

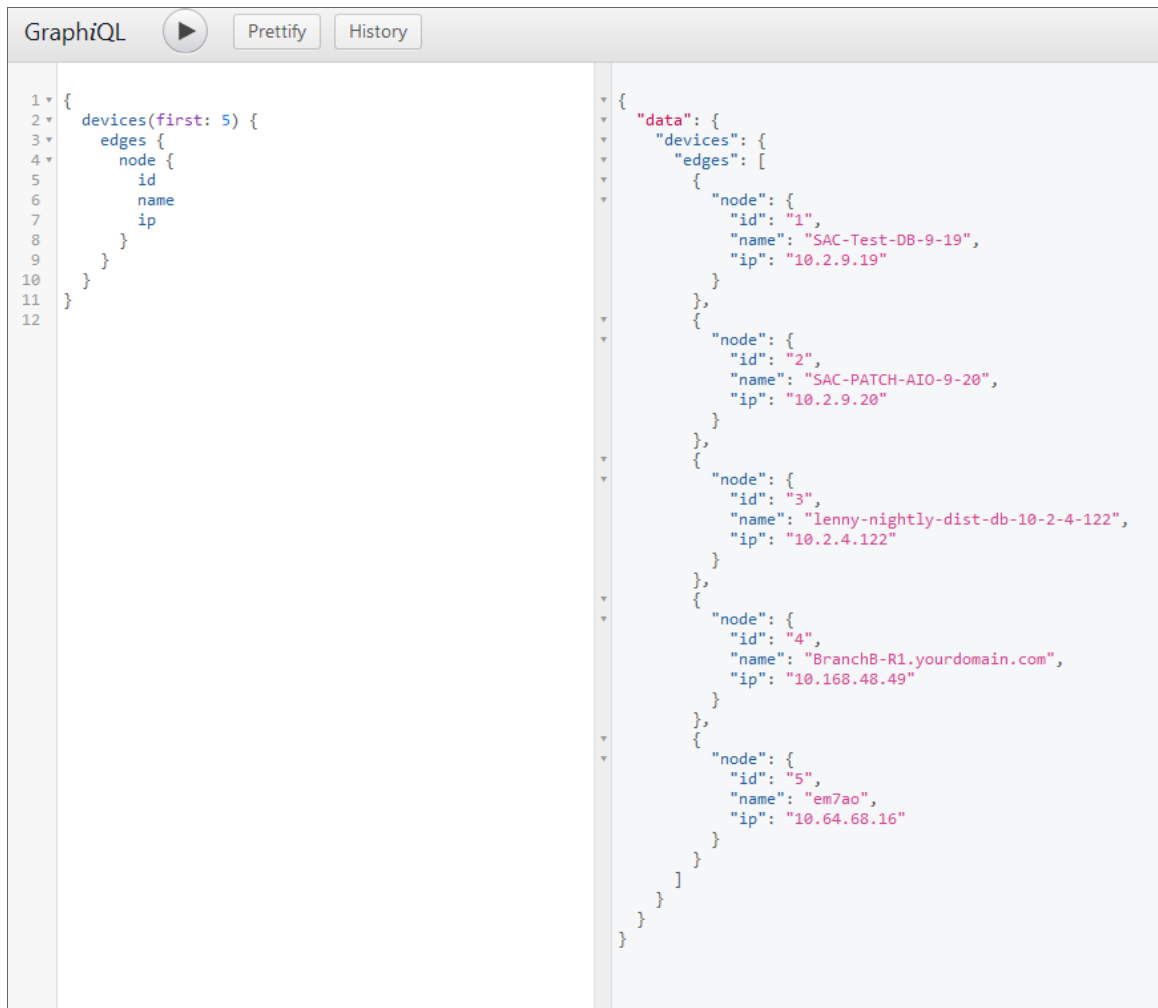
```
query {
  devices (first:5) {
    edges {
      node {
        id
        name
      }
    }
  }
}
```

8. Suppose you want to edit the example query to request the ip address system of each device. The documentation for the "device" object lists "ip" as an available field. In the upper-right pane, add a new line after "name" and type "ip". The GraphQL interface includes auto-complete options when you are typing a query in the upper-left pane. If you type "i" or "ip" on the new line, a list of suggestions appears:

```
1 {  
2   devices(first: 5) {  
3     edges {  
4       node {  
5         id  
6         name  
7         ip  
8       }  
9     }  
10  }  
11 }  
12
```

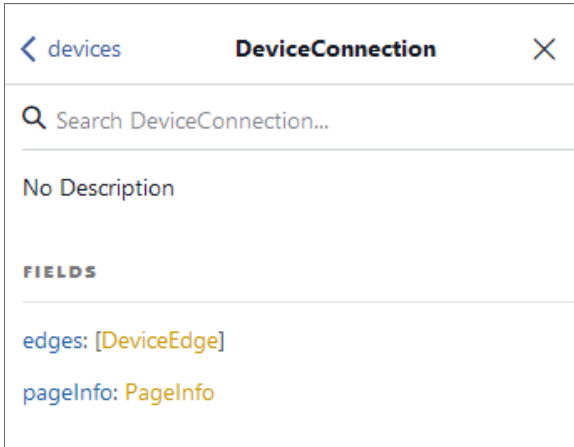


9. After you type or select "ip", click the **[Execute Query]** (Play) button. The query executes and the "ip" field is now included in the result:

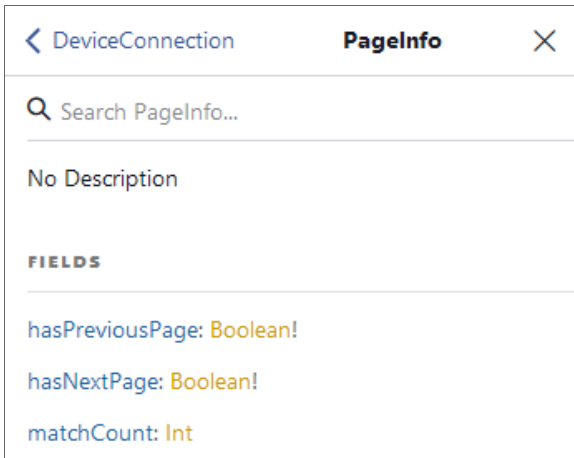


The screenshot shows the GraphQL IDE interface. On the left, a query is written in a monospaced font with line numbers 1 through 12. The query is: `{ devices(first: 5) { edges { node { id name ip } } } }`. Above the query editor are three buttons: a play button (Execute Query), "Prettify", and "History". On the right, the JSON response is displayed, showing a list of five nodes with their respective "id", "name", and "ip" fields. The response is: `{ "data": { "devices": { "edges": [{ "node": { "id": "1", "name": "SAC-Test-DB-9-19", "ip": "10.2.9.19" } }, { "node": { "id": "2", "name": "SAC-PATCH-AIO-9-20", "ip": "10.2.9.20" } }, { "node": { "id": "3", "name": "lenny-nightly-dist-db-10-2-4-122", "ip": "10.2.4.122" } }, { "node": { "id": "4", "name": "BranchB-R1.yourdomain.com", "ip": "10.168.48.49" } }, { "node": { "id": "5", "name": "em7ao", "ip": "10.64.68.16" } }] } } }`

10. Suppose you now want to query for the *next* five device. To do this, you will need to edit the example query to request metadata about the list of values, and then use that metadata to build a second request for the next five devices. In the **Documentation Explorer** pane, navigate back to the "DeviceConnection" object:



11. The "pageInfo" field is the object that defines metadata about the returned list of values, such as the first and last values returned in the list of values in the response. Click "PageInfo" to view how to structure the "pageInfo" section of the query:



The `PageInfo` object contains the following fields:

- **`hasPreviousPage`**. A boolean that indicates whether list entries exist after the last list entry returned by the query.
- **`hasNextPage`**. A boolean that indicates whether list entries exist before the first list entry returned by the query.
- **`matchCount`**. An integer that lists the number of entries matching this query.

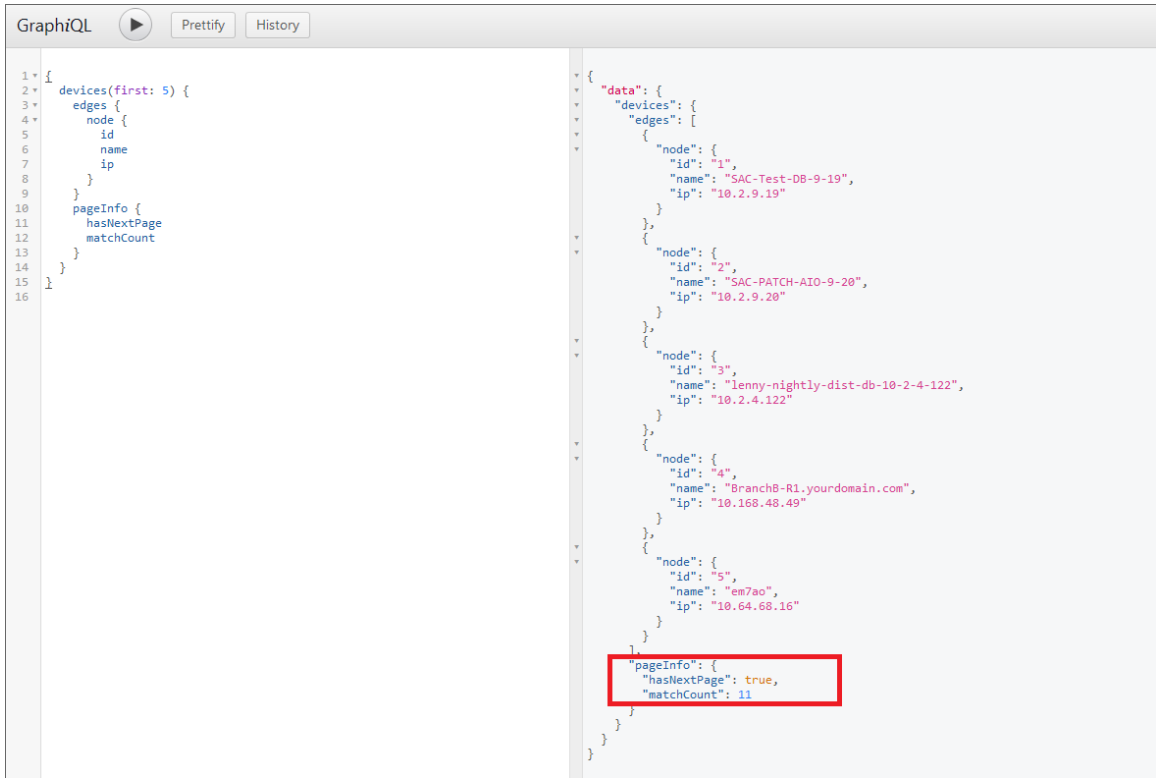
12. Run the following query to add a "pageInfo" section to the example query to request the *hasNextPage* and *matchCount* fields:

```
query {
  devices (first:5) {
    edges {
      node {
        id
        name
        ip
      }
    }
    pageInfo {
      hasNextPage
      matchCount
    }
  }
}
```

The response will include a *pageInfo* section that looks like this:

```
"pageInfo": {
  "hasNextPage": true,
  "matchCount": "11"
}
```

The example response indicates that there are additional devices we can query for beyond the first five, because *hasNextPage* is true, and a total of 11 devices exist according to *matchCount*:



```
1 {
2   devices(first: 5) {
3     edges {
4       node {
5         id
6         name
7         ip
8       }
9     }
10    pageInfo {
11      hasNextPage
12      matchCount
13    }
14  }
15 }
16 }
```

```
{
  "data": {
    "devices": {
      "edges": [
        {
          "node": {
            "id": "1",
            "name": "SAC-Test-DB-9-19",
            "ip": "10.2.9.19"
          }
        },
        {
          "node": {
            "id": "2",
            "name": "SAC-PATCH-AIO-9-20",
            "ip": "10.2.9.20"
          }
        },
        {
          "node": {
            "id": "3",
            "name": "lenny-nightly-dist-db-10-2-4-122",
            "ip": "10.2.4.122"
          }
        },
        {
          "node": {
            "id": "4",
            "name": "BranchB-R1.yourdomain.com",
            "ip": "10.168.48.49"
          }
        },
        {
          "node": {
            "id": "5",
            "name": "em7ao",
            "ip": "10.64.68.16"
          }
        }
      ]
    },
    "pageInfo": {
      "hasNextPage": true,
      "matchCount": 11
    }
  }
}
```

Querying the ScienceLogic GraphQL API from an External Application

After you have determined the queries you want to execute using the GraphQL interface, you can execute those queries from an external application by performing an HTTPS request to the GraphQL URI for your SL1 system. To execute a GraphQL query using an HTTPS request:

- Use the POST method.
- Set the content-type header to "application/json".
- Specify the username and password of an appropriate user account.
- In the request content, send JSON in the following structure:

```
{
  "query": "[GraphQL query]"
}
```

For example, to execute the original example query, you would POST the following JSON:

```
{  
  "query": "query devices { devices { edges { node { id 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



ScienceLogic

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

International: +1-703-354-1010