DataStax Studio 6.7 (Latest version)

Updated: 2019-01-19-08:00

Table of Contents

About DataStax Studio 6.7

DataStax Studio is an interactive developer tool for CQL (Cassandra Query Language), Spark SQL, and DSE Graph. Developers and analysts collaborate by mixing code, documentation, query results, and visualizations in self-documenting notebooks.

- For CQL *(page 21)*, use DataStax Studio to visually create and navigate database objects, create complex queries, and tune CQL queries. The schema helps you visualize the keyspaces in a tree-like view.
- For DSE Graph, use DataStax Studio to explore and view large datasets. The code for DSE Graph is written in the Gremlin graph traversal language and is executed by the Gremlin Server that is a part of the DSE Graph component. The graph schema view *(page 15)* helps you visualize the graph organization and connections.
- For DSE Analytics, write, test, and run Spark SQL queries *(page 22)* against DSE clusters.
- The intelligent code editor provides syntax highlighting, validation, intelligent code completion, configuration options, and query profiling.
- Self-documenting notebooks *(page 5)* mix runnable code, documentation (markdown text), and visualizations for query results. This rich interactive environment provides an intuitive interface for developers and analysts to collaborate and test theories. Markdown is a simple language for creating human-readable plain text.

See:

- Installing DataStax Studio 6.7
- Upgrading DataStax Studio 6.0 or 6.7

DataStax Studio security

DataStax Studio does not provide a multi-user experience. If deployed in a central fashion, grant only to users who need to see and edit the notebooks and the database connections that are associated with that DataStax Studio instance.

Access to DataStax Studio is not protected by built-in authentication. Exposing direct access to DataStax Studio outside of a host-local network interface is strongly discouraged.

> **Important:** Changing the httpBindAddress setting from the default (localhost) can pose a security risk as users on external machines can gain access to notebooks and the DSE clusters those notebooks are connected to. Studio is designed to be used as a desktop application. Distributed deployment introduces potential security risks.

About DataStax Studio notebooks

Notebooks provide an intuitive interface for developers and analysts to collaborate by mixing code, documentation, and visualizations for query results. A notebook consists of one or more cells.

Notebooks combine a web browser user interface with the DataStax Enterprise scalable database. Notebooks provide a sharable interface to access data and enable collaboration, sharing, and explanation.

Notebooks contain the inputs and outputs of an interactive session and descriptive text that accompanies the code.

DataStax Studio 6.0 provides these notebooks:

What's New in Studio v6.0.0
: Overview of new features: notebook export and import, interactive graph with neighborhood expansion, notebook revision history, and Spark SQL to write and submit Spark SQL queries against DSE Analytics.

Working with CQL
: Tutorial to create and interact with data in a DSE cluster by writing and executing CQL code in a notebook. Learn about viewing the CQL schema with DESCRIBE commands. Learn to use content assist, syntax validation, and domain validation. View results in table view or different styles of charts. See also CQL documentation.

Working with Graph
: Introduces the DSE Graph-centric features of Studio and provides a basic introduction to vertices sizes, edge colors, and using interactive graph *(page 15)* features. Learn how to:

  - Use display name templates to improve the display names shown in graph view
  - Dynamically view results in multiple views such as JSON, table, charts, and graph
  - Examine a subset of a graph based on an edge label, a set of edge labels, or a set of edges
  - Select vertices and edges, learn about vertices sizes and colors
  - Use full screen
  - Hide the code editor
  - Use content assist and schema-aware content assist for Gremlin code
  - Use schema view to see what is connected and how

  Learn about charting to visualize data results. Learn how to view help for efficient keyboard shortcuts *(page 40)*.

Working with Spark SQL v6.0.0
: Quick tips and brief tutorial for working with Spark SQL in Studio. Learn to use content assist while writing a Spark SQL statement. Use schema view for a tree view of schema elements in a database. If you are new to DSE Analytics, see DSE Analytics.

DSE Graph QuickStart
: Introduction tutorial for using Studio to learn about DSE Graph. The tutorial provides steps to insert data and run traversals. Also, see the DSE Graph QuickStart documentation.

DataStax Studio 6.7 release notes

Release notes for DataStax Studio 6.7.x.

For DSE release notes, see DataStax Enterprise 6.7 release notes.

For upgrade information, see Upgrading DataStax Studio.

DataStax Studio 6.7.0 release notes

5 December 2018

The latest version of DataStax Studio is 6.7.0.

DataStax Studio release notes for 6.7.0 include:

6.7.0 Changes and enhancements

- OpenJDK 8 (1.8.0_151 minimum) is required. DataStax does more extensive testing on OpenJDK 8 due to the end of public updates for Oracle JRE/JDK 8. (STUDIO-2309)
- Studio startup time is improved. (STUDIO-2406)

6.7.0 Resolved issues

- Exported notebooks include results sets even when `cell code only` is selected. (STUDIO-2335)
- Studio bash script error when checking Java version on Ubuntu 16. (STUDIO-2410)
- Studio hangs due to race condition during a connection exception. (STUDIO-2451)

Installing DataStax Studio 6.7

Steps for installing and running DataStax Studio 6.7.

See Installing DataStax Studio 6.7.

Upgrading DataStax Studio 6.7

Upgrading to DataStax Studio 6.7.

See Upgrading DataStax Studio 6.7.

Using DataStax Studio 6.7

Starting and stopping DataStax Studio

Prerequisites: Studio requires OpenJDK 8 or Oracle Java 8 (at least 1.8.0_151). Verify that the required Java version is used. See Studio will not start with wrong Java version.

Steps to start and stop Studio.

1. To start Studio, run the Studio Server shell script:
   - Linux:

     ```
     $ cd installation_location/datastax-studio-6.0.0
     ```

     ```
     $ ./bin/server.sh
     ```

     Tip: To start Studio in the background, add & at the end of the command:

     ```
     $ ./bin/server.sh &
     ```

   - Windows:

     ```
     C:/> installation_location\datastax-studio-6.0.0\bin\
     ```

     ```
     C:/> server.bat
     ```

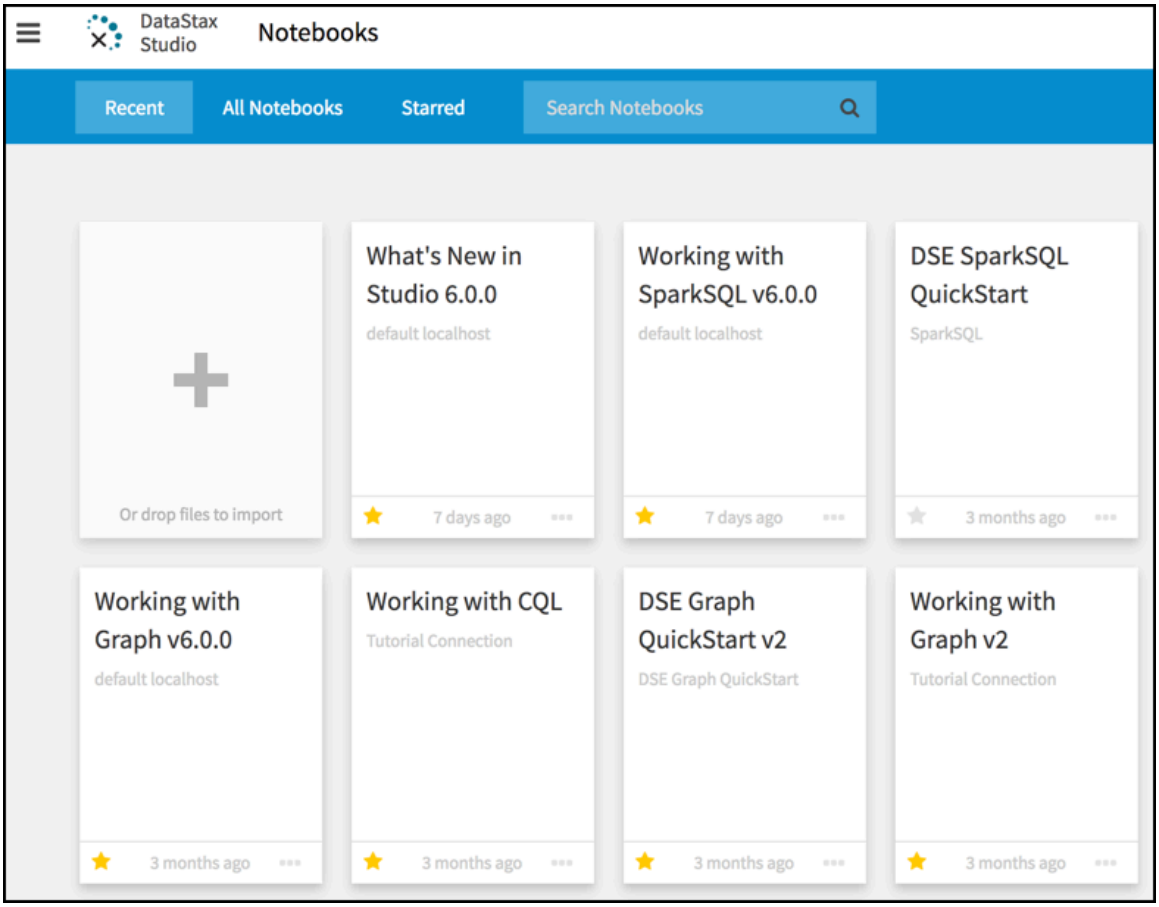
   Your result will look similar to:

   ```
   Studio is now running at: http://127.0.0.1:9091
   ```

2. To run DataStax Studio, enter this URL in your web browser:

   `http://URI_running_DSE:9091/`

   - For DSE running on localhost, *URI_running_DSE* is localhost.
   - For DSE on another machine, *URI_running_DSE* is the hostname or IP address for the remote machine.



3. To stop Studio:

```
$ pkill -f studio
```

Creating a connection in DataStax Studio

A connection from each notebook to a DSE cluster is required. A notebook persists its data to a DSE cluster.

Connect notebooks only to DSE clusters that run the relevant workload. For example:

- To run Gremlin cells, create a connection to a DSE cluster that is configured for and running DSE Graph.
- To run AlwaysOn SQL queries in Studio, create a connection to a DSE cluster that is configured for and running the AlwaysOn SQL service.

Each notebook has only one connection. A connection can serve multiple notebooks.

Tip: You can configure more than one host in a Studio connection. Hosts initialize the Cassandra driver connection, so more than one host provides redundancy and failover protection.

Prerequisites: The DSE cluster must be installed and running.

To create a connection from a notebook to a DSE cluster:

1. Browse to the URL for the Studio installation:

   ```
   http://running_DSE:9091/
   ```

2. In the menu (#), select **Connections**.

   The existing connections are displayed in a list. The list is sorted in alphabetical order and shows the port and host for each connection.

3. Click **+** in the top center of the page.

4. In the **Create Connection** dialog, enter the connection information:

   Name
   : Name of the connection from the notebook to a DSE cluster.

   Host/IP (comma delimited)
   : The host names or IP addresses of the DSE cluster to connect to. Default: localhost.

   Username
   : Optional. DSE username for logging in.

   Password
   : Optional. DSE password for logging in.

   Port
   : IP connection port. Default: 9042.

   For example, to connect to a single-node DSE cluster on the local host using the default port.

   Name
   : My First Connection

   Host/IP
   : `127.0.0.1`

   Port
   : 9042

5. Click **Test** to verify the connection works.

6. To configure a secure encrypted connection, select the **Use SSL** check box.

The Truststore and Keystore fields display. See Using SSL connections in DataStax Studio *(page 13)*.

7. Click **Save**.

   A connection is created from the notebook to the DSE cluster.

Using SSL connections in DataStax Studio

Prerequisites:

SSL must be configured and working on your DSE cluster. See Configuring SSL for details on:

- Client-to-node encryption
- Node-to-node encryption
- Preparing server certificates

DataStax recommends using certificates signed by a CA. The truststore or CA certificate can be shared between all DSE servers and clients. See Using local SSL certificate and keystore files.

For DataStax Studio, the public key certificate from the CA must be stored in a local truststore file.

To perform server verification, the client needs to have the public key certificate of each node in the cluster stored in a local truststore file. This truststore file is password protected.

1. In the menu (#), select **Connections**.
2. Click **+** to add a connection or click any of the listed connections to edit an existing connection.
3. In the **Create Connection** dialog, enter the connection information:

   Name
   Name of the connection from the notebook to a DSE cluster.
   Host/IP (comma delimited)
   The host names or IP addresses of the DSE cluster to connect to. Default: localhost.
   Username
   Optional. DSE username for logging in.
   Password
   Optional. DSE password for logging in.
   Port
   IP connection port. Default: 9042.

   For example, to connect to a single-node DSE cluster on the local host using the default port.

   Name

My First Connection

Host/IP

`127.0.0.1`

Port

9042

4. Select the **Use SSL** check box to show the Truststore and Keystore fields.
5. As appropriate for your environment, enter the local file path and password for the truststore.
6. Click **Test** to verify the connection to the DSE cluster.

Using graph in DataStax Studio

Graph databases are useful for discovering simple and complex relationships between objects. Relationships are fundamental to how objects interact with one another and their environment. Graph databases perfectly represent the relationships between objects.

Using DSE Graph in DataStax Studio provides an interactive way to:

- Compose Gremlin traversals and view results in multiple dynamic views such as JSON, table, charts, and graph
- Leverage Studio's schema view and schema-aware content-assist for rapid Gremlin Traversal development
- Manipulate the graph view with interactive graph
  # Expand vertex neighborhoods
  # Remove vertices and edges from view

For an interactive introduction to working with DSE Graph in Studio, see the DSE Graph notebook tutorials that are installed with Studio:

Working with Graph

Introduces the DSE Graph-centric features of Studio and provides a basic introduction to vertices sizes, edge colors, and using interactive graph *(page 15)* features. Learn how to:

- Use display name templates to improve the display names shown in graph view
- Dynamically view results in multiple views such as JSON, table, charts, and graph
- Examine a subset of a graph based on an edge label, a set of edge labels, or a set of edges
- Select vertices and edges, learn about vertices sizes and colors
- Use full screen
- Hide the code editor
- Use content assist and schema-aware content assist for Gremlin code
- Use schema view to see what is connected and how

Learn about charting to visualize data results. Learn how to view help for efficient keyboard shortcuts *(page 40)*.

DSE Graph QuickStart

Introduction tutorial for using Studio to learn about DSE Graph. The tutorial provides steps to insert data and run traversals. Also, see the DSE Graph QuickStart documentation.

The Studio graph view provides a contextual view of a result by automatically populating the entire sub-graph shared between the result's vertices and edges. The Gremlin result is examined:

- If a result contains vertices, then all edges linking those vertices are automatically populated.
- If a result contains edges, then the adjacent vertices are automatically populated.
- If the result is a subgraph, then it is displayed as-is.

Path results can contain arbitrary Java objects (not just vertices and edges), so path results are not rendered by the graph view.

Using interactive graph in DataStax Studio graph view

Studio provides these interactive graph features:

- Expand vertex neighborhoods, with the ability to filter by edge label
- Remove vertices and edges from the graph view

Tip: Right-click vertices and edges to activate these features.

In a gremlin cell, you can interactively expand the neighborhood in graph view.

Tip: At any time, double-click the graph view to recenter the display.

Edge population in neighborhood expansion happens in two stages:

1. Adjacent edges and their corresponding other vertices are retrieved.
2. Edges between new vertices and existing vertices may be automatically populated based on the selected Edge Population Strategy.

Neighborhood expansion has configurable view options:

- Populate all common edges, the default complete view.
- Populate only the edges of the selected labels to expand.

  All edges that connect a newly retrieved vertex and an existing vertex are added to the view only if their EdgeLabel is selected.

- Do not auto-populate common edges.

Open and use the Working with Graph notebook to learn more about neighborhood expansion.

Notebook code editor in Gremlin code cells

The Notebook code editor supports the Apache Groovy™ programming language in Gremlin code cells. Syntax validation supported for a subset of Groovy.

A notebook connects to a DSE cluster based on the connection information. Each notebook has only one connection. One piece of information that a connection specifies is its **Graph Name**.

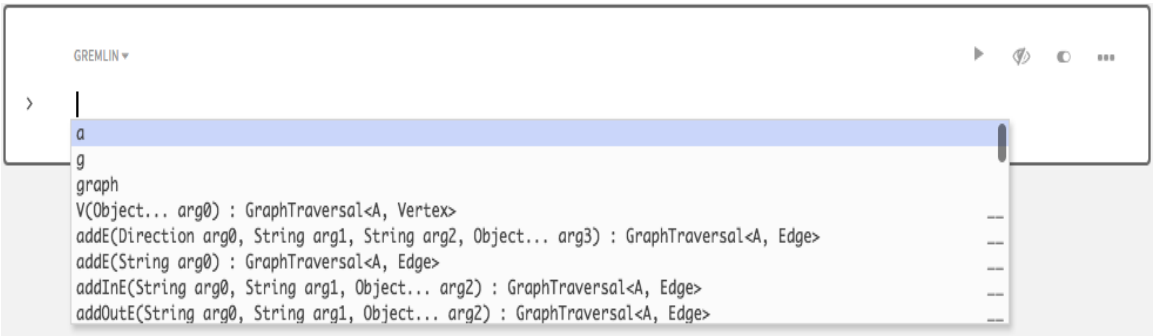
The code editor provides pre-defined alias variables:

graph
: The variable that refers to the graph.

g
: The traversal source associated with the graph for OLTP graph traversals.

a
: The traversal source associated with the graph for OLAP graph traversals.

A drop-down list shows for content-assist *(page 17)* on a blank line in the editor.



The code editor supports inline code validation. If some code is invalid or uses Groovy syntax that the editor does not support, the invalid code is displayed with a red underlining.

Use the slider () to Disable editor validations to turn off.

Cancel execution

When code is running in a Gremlin cell, you can cancel the execution to kill the gremlin session. This cancel execution is helpful when you want to make a change to the code and do not want to wait until a long running query finishes.

- Click Kill Session.

  The confirmation dialog appears:

```
Gremlin cell executions will be stopped by killing the notebook
 session. Proceed?
```

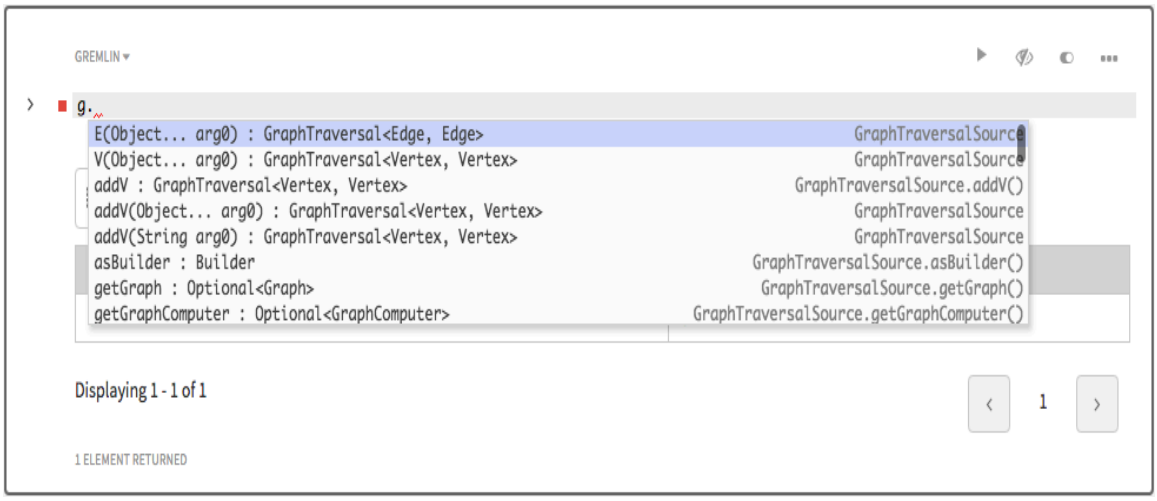
- Click Yes to proceed, no to cancel.
- After you cancel execution, this message is shown:

```
The Gremlin Session associated with the notebook was killed.
```

- You can make code changes to the cell and run the execution again.

Content assistance

The code editor provides content assistance (**Ctrl+Space**).



Content-assistance proposes content based on context:

- Methods
- Variables

Press **Return** to select the highlighted choice from the assistance list.

You can filter proposals by typing enough characters to select a line in the pull-down menu.

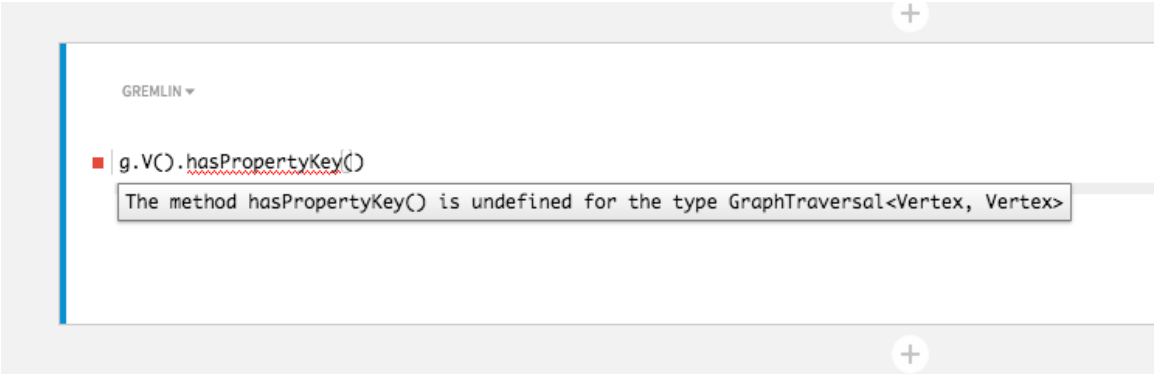
Use Shift-Tab to return to the beginning of a line.

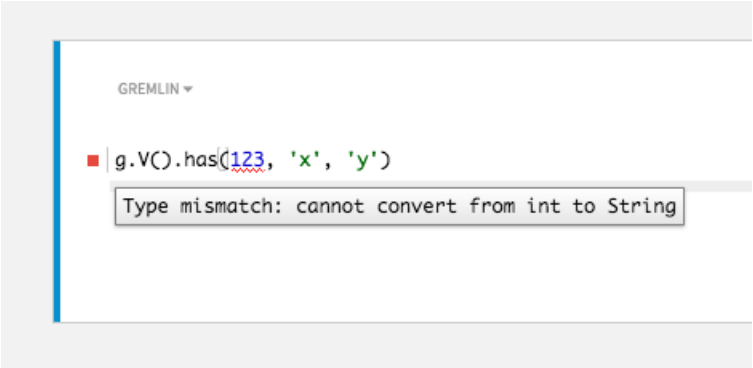
Code validation

The notebook editor supports these validations:

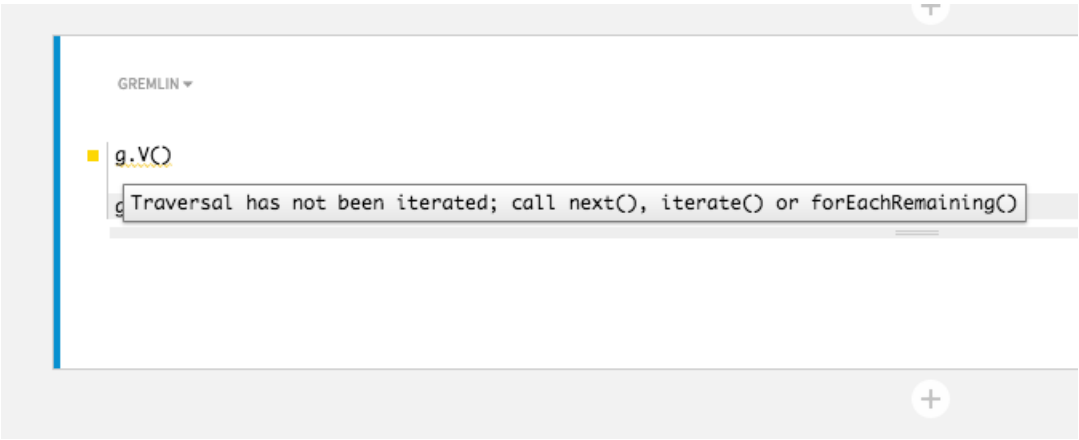
- Groovy syntax—The code in a Gremlin cell is executed within DSE Graph as Groovy. The notebook editor adds enough Groovy syntax support to help you craft Gremlin statements.

- Type-checking—The code in a cell is also checked for type. If you try to call a method on an object of the wrong type or pass a parameter of an invalid type, a validation error is displayed.





- Domain-specific—Common errors in code are pointed out when possible.



The notebook editor performs validation as code is entered into the cell. You can turn off validation on a per-cell basis.

The code editor cells in a notebook have no Gremlin session scope context and rely on an implicit scope based on their order. Validation occurs from the top cell down in the order in the notebook. If you execute code out of order, validation errors can occur even if the code executes successfully.

Groovy language support

Supported:

- variable declarations in Groovy style:
  ```
  # def foo
  # def SomeType foo
  # SomeType foo
  # Shell-style variables
  ```
- method invocations
- optional semicolons to complete statements
- generics
- strings in both forms:
  ```
  # '123'
  # "123"
  ```
- `for` loops: basic and for in syntax
- `while` loops
- `try-catch-finally`: a type is required for catching the exception
- `if-else` statements
- `switch` statements
- casting in both styles: `(SomeType) foo` and `foo as SomeType`
- list literals

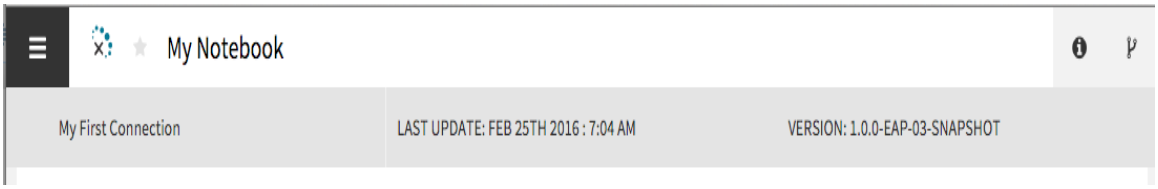
Unsupported Groovy language

- closures
- multi-variable assignment and other advanced variable assignments like object deconstruction
- `import` statements explicitly disabled
- `map` literals or other complex type literals that are not supported by Java
- `try-catch-finally`: multi-catch is not supported

Notebook information

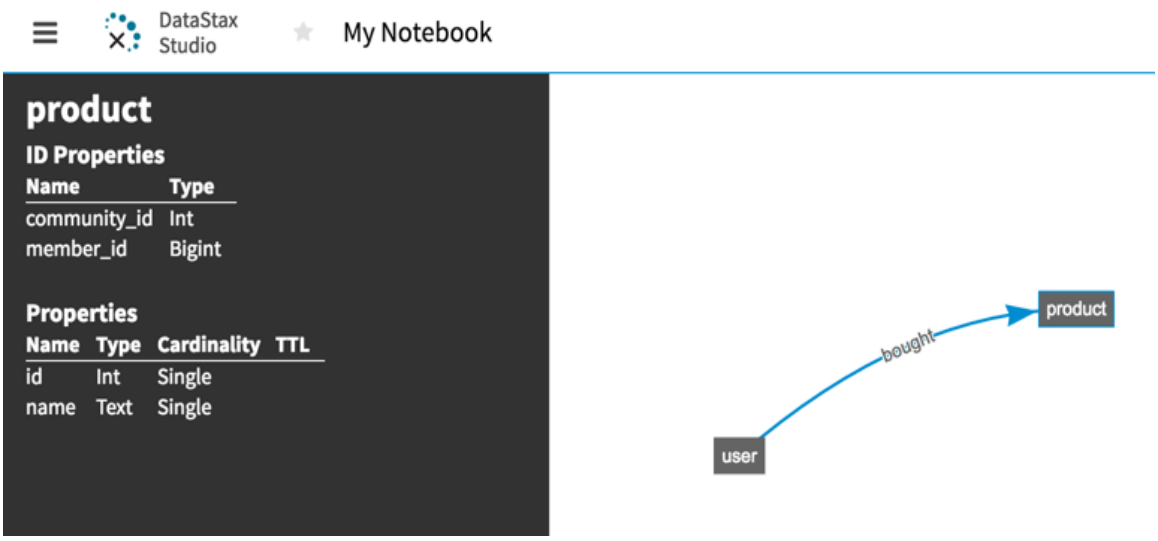
Selecting **Information** displays:

- Connection being used for the notebook.
- Last update to the notebook.
- Version of Studio JAR file being run.



Notebook schema

Selecting **Information** displays the current graph schema for the notebook.



In addition to visual schema representation, the code-assist feature also provides schema-assist proposals.

That's a lot of proposals. Let's break them down:

- The editor is intelligent enough to know that this is a Vertex-based traversal because of type inference. The editor presents only the schema proposals that are relevant for vertex properties and keys.
- Because there are multiple `has` methods, Studio proposes possibilities for all of these variations.
- The editor concretely makes a proposal for each possible property key.

Using CQL in DataStax Studio

The notebook tutorial **Working with CQL** is installed with Studio. The tutorial provides hands-on steps to create and interact with data in a DSE cluster by writing and executing CQL code in a notebook. This interactive notebook provides interactive steps to use CQL in DataStax Studio to:

- Select CQL as the language to enable intelligent editor features
- View the CQL schema
- Use CQL templates and content assist to help you form valid CQL statements:
  # Propose valid CQL keywords
  # Propose valid table names anywhere a table can be referenced in a CQL statement
  # Propose column names anywhere a table can be referenced
- Identify errors with DSE domain-specific validations when CQL statements violate a constraint
- View results in table view or different styles of charts

You can use the keyspace menu to select the keyspace or specify the CQL USE statement. The following rules apply to USE statements in Studio:

- USE statements apply only to the current cell.
- A USE statement overrides the default keyspace setting for all statements that follow the USE statement within that cell.
- The keyspace selected in the keyspace drop down menu applies only to the current cell.

- The selected keyspace is copied to new cells created below an existing CQL cell.
- If a default keyspace is not selected, and there are no USE statements in the cell, all statements must be fully qualified to include the desired keyspace.

See the DataStax dev blog post Studio Multi-Model CQL Support.

Using Spark SQL in DataStax Studio

Analyze data stored in DSE clusters with Spark SQL relational queries. Spark SQL is a unified relational query language for traversing over distributed collections of data, and supports a variation of the SQL language used in relational databases.

Spark SQL notebook features include:

- Interactively perform Spark SQL queries against a DSE cluster
- Schema-aware content assist
- Syntax validations to facilitate faster prototyping

To run Spark SQL queries in Studio:

- The DSE cluster must be configured for the AlwaysOn SQL service.
- Be familiar with the Supported syntax of Spark SQL.

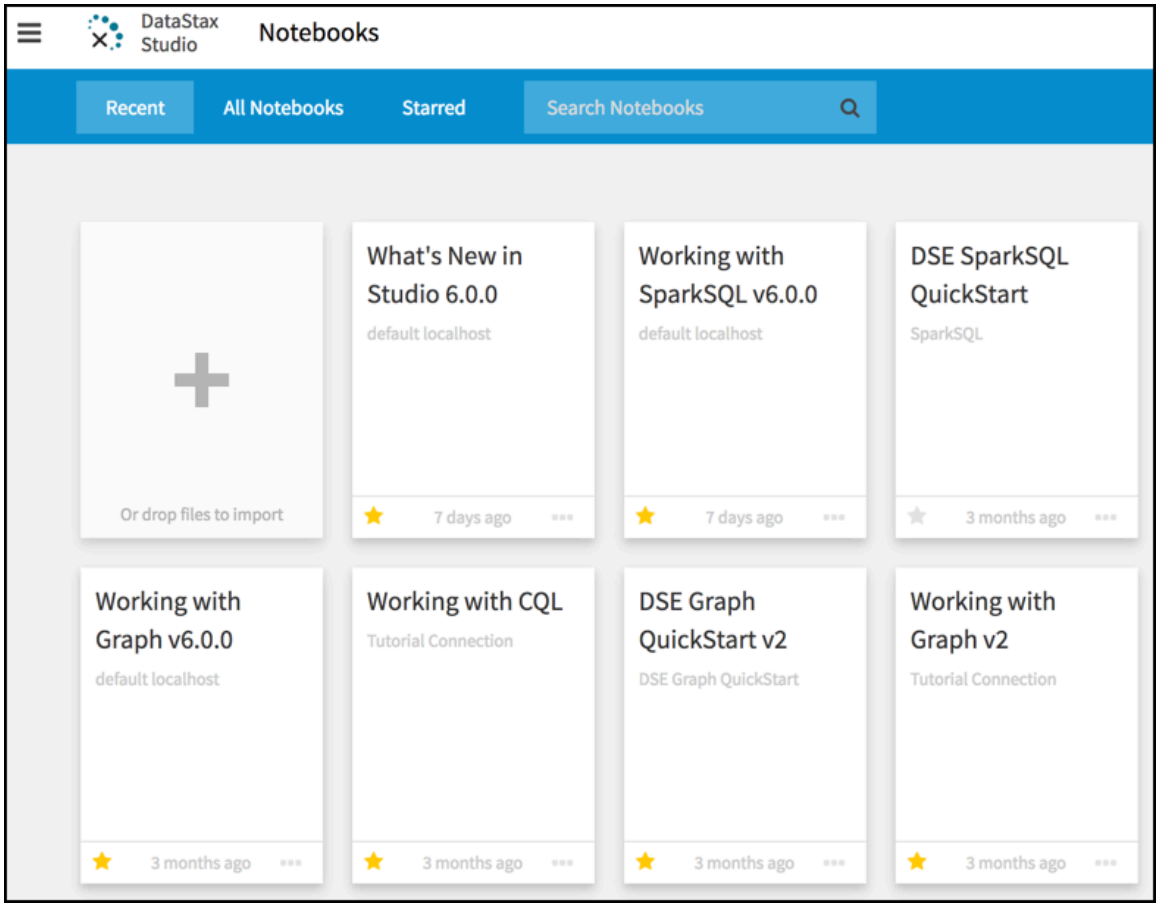
If the AlwaysOnSQL service is turned on, Studio uses the JDBC interface to pass queries to DSE Analytics. Two tables, *graphName_vertices* and *graphName_edges*, are automatically generated in the Spark database *dse_graph* for each graph, where *graphName* is replaced with the graph used for the Studio connection assigned to a Studio notebook. These tables can be queried with common Spark SQL commands directly in Studio, or can be explored with the dse spark-sql shell. To learn more about using Spark SQL to query, see the Using Spark SQL to query data documentation.

The notebook tutorial *Working with SparkSQL v6.0.0* is installed with Studio. The tutorial provides hands-on steps to create data and execute Spark SQL code in a notebook. Learn about exploring the SQL schema in schema view, using content assist for syntax and domain validation. View results in table view and different styles of charts.

Listing notebooks in DataStax Studio

The Notebook Manager lists all notebooks, starred notebooks, or filtered notebooks.

1. Open the Studio server by entering `http:/URI_running_DSE:9091/` in your web browser.



DataStax Studio opens to the Notebooks page.

2. To filter notebooks, click:
   - All Notebooks
   - Starred
   - To filter by notebook name, start typing text in the Search Notebooks field.

   The default notebook filter is Recent.

3. To return to the Notebook Manager from a notebook, select **Notebooks** from the menu (#).

Using notebook history

Notebook revisions provide a self-describing persisted state of a notebook. Major changes, like executing a cell and getting a new result, create a new revision immediately.

Changes to a notebook are automatically tracked with notebook revisions that provide a historical dated record with descriptions and change events that support easy retrieval. You can provide revision names and short comments to specific persisted states of a notebook to identify specific milestones in the history of a notebook. You can filter and retrieve notebook

revisions. Notebook revisions are the historical versions of a notebook. When viewing notebook history, the notebook preview is read-only.

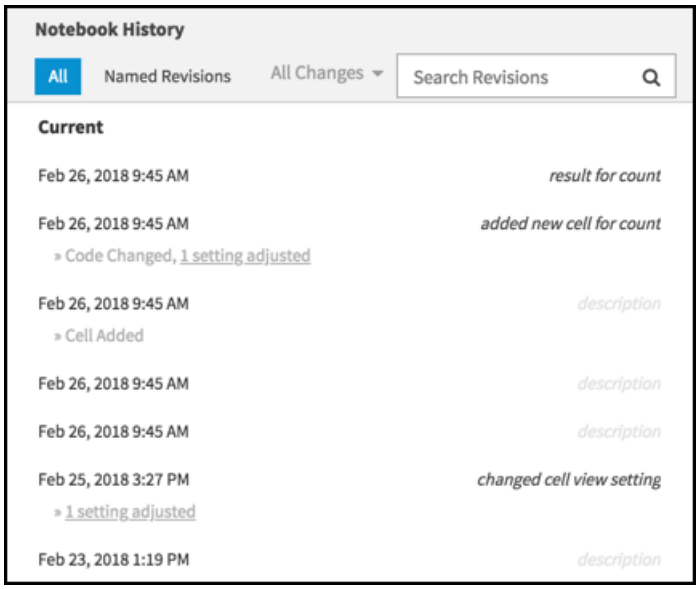
Events are saved only when the cell is executed. Automatic notebook revisions are saved before these events:

Event	Description
Cell Added	A cell is added to a notebook.
Cell Deleted	A cell is deleted from a notebook.
Cell Language	The cell language is changed: Markdown, Gremlin, CQL, or Spark SQL.
Cell Schema	A schema is created or modified.
Cell View	Changes are made in a cell with returned data.
Cell View Settings	The view settings are changed in a cell with returned data.
Code Changed	Code in a cell is changed.
Gremlin Execution Engine	The run configuration is changed.
Notebook Reverted	A notebook is reverted.

1. In any notebook, click (View History) in the upper-right corner.

   While viewing notebook history, the notebook is read-only.

2. The **Notebook History** panel displays on the right. The default view is to show all revisions in descending date order.

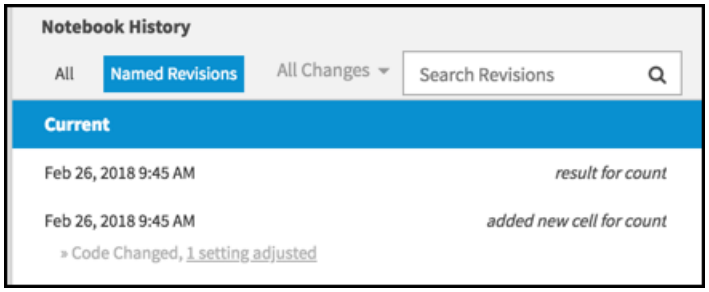


3. To view revisions for a specific change event, click **All Changes** #, and select a change event.

   For example, to view only the change events where code in a cell was changed, select **Code Change** #.

4. To view all changes again, click **Code Change** # and select **All Changes** from the list of change events.

5. To view only named revisions, click **Named Revisions**.



   All filters apply. For example, to view only named revisions with code changes, apply the **Code Change** filter and then click **Named Revisions**.

6. To filter named revisions, enter text in the Search Revisions text box.

   The results are filtered as you type.

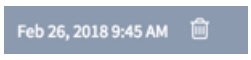
7. To view a notebook revision, select a revision in the **Notebook History** panel.

   The notebook preview is read-only.

8. To restore a notebook revision, click **Restore Revision**.

   The notebook history is updated.

9. To delete a notebook revision, hover over the revision and click .



10. A notebook revision is view-only. To update a notebook, click Cancel.

Defining run behavior with execution configurations

With results visualization and profiling capability, Studio is a powerful debugging tool by executing code in a way that reproduces the settings used in their applications. When using Studio to interact with DSE, you are able to execute code written in CQL, Gremlin, or Spark

SQL. Each cell language has its own set of configuration options that determine execution behavior.

Execution configurations provide different execution setups by passing a set of options that customize the run execution. Execution configurations are persistent so you can reuse them.

Standard run configurations are provided:

Cell type	Execution configuration	Settings
CQL	CL.ONE	• Consistency level: ONE • Timeout (MS): 10000 • Max Results: 5000 • CQL Tracing: Disabled
CQL	CL.Quorum	• Consistency level: QUORUM • Timeout (MS): 10000 • Max Results: 5000 • CQL Tracing: Disabled
CQL	CL.ALL	• Consistency level: ALL • Timeout (MS): 10000 • Max Results: 5000 • CQL Tracing: Disabled
CQL	CL.ONE.TRACE	• Consistency level: ONE • Timeout (MS): 10000 • Max Results: 5000 • CQL Tracing: Enabled
Gremlin	Transactional	Execute statement using real-time (transactional) engine for OLTP (online transaction processing), traversal source `g`.
Gremlin	Analytic	Execute statement using analytic engine (Spark) for OLAP (online analytical processing), traversal source `a`.
Spark SQL	Run	Execute statement with Spark SQL query engine with AlwaysOn SQL service.

You can create a custom CQL execution configuration by adjusting these settings for your environment:

- Consistency level: ANY, ONE, TWO, THREE, QUORUM, ALL, LOCAL_QUORUM, EACH QUORUM, SERIAL, LOCAL_SERIAL, LOCAL_ONE
- Timeout (MS): *`milliseconds`*
- Max Results: *`number of records`*

- CQL Tracing: Enabled or Disabled

1. In the upper-right corner of a code cell, hover over the play button # and then click # to list the existing configurations.

   Tip:

   - In CQL cells, the default execution configuration is CL.ONE so the play button is **CL.ONE** #.
   - In Gremlin cells, the default execution configuration is Execute using real-time (transactional) engine so the play button is **Real-time** #.
   - In Spark SQL cells, all statements are executed with Spark SQL query engine with AlwaysOn SQL service.

2. Select **Manage Configurations**.
3. Select an existing configuration to view settings.
4. **For CQL cells only:** Click **+ Add New Configuration** to create an execution configuration.
   a. Enter a self-describing **Name**.
   b. Select a **Consistency Level**.
   c. Enter a **Timeout** value in milliseconds.
   d. Enter **Max Results**.
   e. Select the **CQL Tracing** check box to enable CQL tracing.
5. Click **Save**.

   New execution configurations are available to be selected from any CQL notebook cell.

Exporting notebooks in DataStax Studio

Export a notebook to a packaged compressed file. Options for notebook export are:

- Cell code and result sets

  Export the entire notebook. Useful for collaborating.

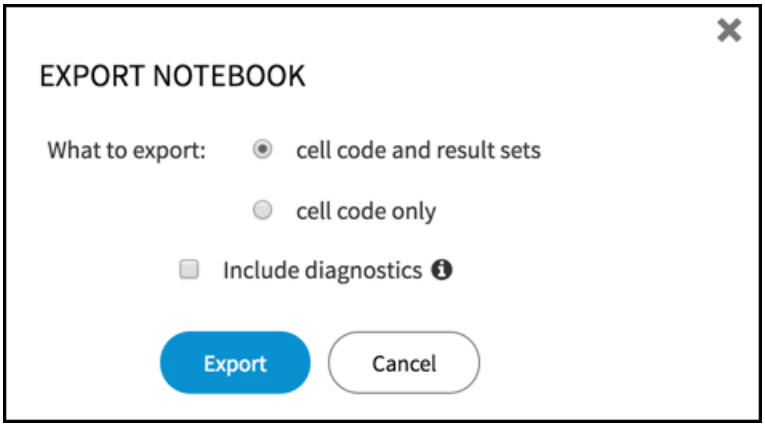
- Cell code only

  Export only the cell code of the notebook. Useful for creating a clean copy of the notebook. For example, for a tutorial.

- Include diagnostics

  Export the notebook with diagnostic data for troubleshooting purposes. May include sensitive Information. Credentials are not included.

1. To export a notebook:
   - In any open notebook, in the menu (#), select **Export this Notebook**.
   - On the Notebook Manager page, click the ellipsis button (**…**) to show more actions for the notebook you want to export, and click **Export**.



2. Select export options and click **Export**.

   As appropriate, select to include cell code and results or cell code only. For either of these actions, you can also select to include diagnostics. Exporting a notebook with diagnostics might include sensitive Information. Credentials are not included in the export.

   The notebook is exported to the downloads location as a packaged compressed file. For example: DSE_SparkSQL_QuickStart_6.0.0_2018-02-28_15_03_41.studio-nb.tar.

3. Share the exported notebook using any file manager.

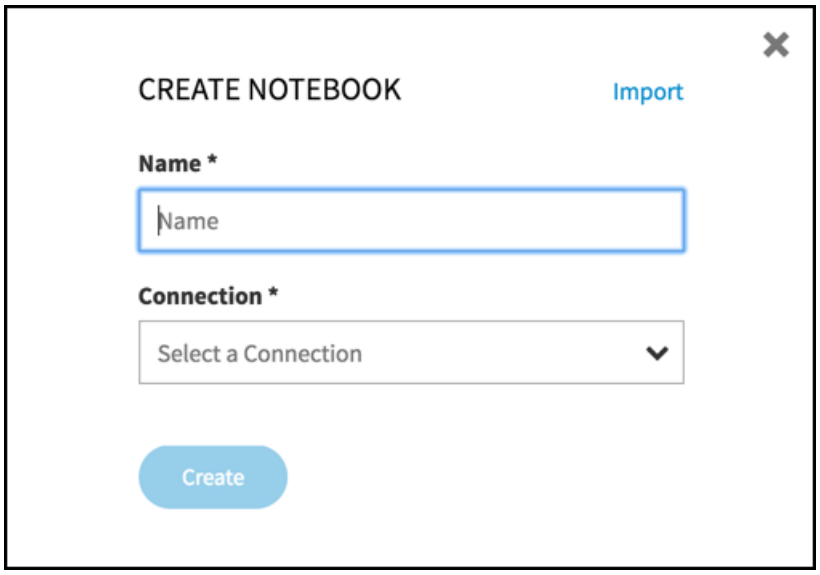
Importing notebooks in DataStax Studio

You can import a notebook using drag and drop or with the file system. The contents of the imported notebooks are defined during the export process. For example, a notebook that was exported with cell code only will be imported without result sets. If a notebook with an existing name is imported, duplicate notebook names are renamed (1), (2), and so on.

1. Open the Notebook Manager page.
2. To import a notebook by dropping a packaged compressed file, open a file manager and drag the file to the Notebook Manager page.
3. To import a notebook using the Import dialog:

a. Click **+** on the empty notebook cell.

b. Click **Import** on the Create Notebook dialog.



c. Navigate the file manager and select the notebook you want to import.

4. Select a connection for the notebook you are importing. Click **Save**.

   If the notebook is a graph notebook, the option to specify a graph is available if the connection is up for a graph-enabled DSE cluster.

5. The notebook is imported with the connection and graph. To change the connection or graph, select **Configure this Notebook** in the menu (#).

Configuring DataStax Studio 6.7

Studio is a desktop application that automatically saves your work. The default configuration should not require adjusting.

Configuring DataStax Studio

Studio is a desktop application that automatically saves your work. The default configuration should not require adjusting. However, you can configure different aspects of how Studio runs, including changing the httpBindAddress.

> **Important:** Changing the httpBindAddress setting from the default (localhost) can pose a security risk as users on external machines can gain access to notebooks and the DSE clusters those notebooks are connected to. Studio is designed to be used as a desktop application. Distributed deployment introduces potential security risks.

To change JVM settings, see Specifying JVM settings for DataStax Studio *(page 34)*.

1. Open the `configuration.yaml` file in a text editor.
2. Make changes to any basic *(page 30)* setting.
3. If an advanced *(page 33)* setting is not in the file, add it.
4. Save the file.
5. Restart Studio to recognize the changes.

configuration.yaml Studio configuration file

The configuration file for Studio is *`dse-studio-install-dir`*`/configuration.yaml`.

A sample configuration file in XML format:

```
# Maximum number of items returned per cell execution. Additional items
 will be truncated.
# Unit: count / number of items
resultSizeLimit: 1000

# Maximum size of a cell result. If a cell result exceeds this size then
 the cell execution will fail.
# Unit: bytes
maxResultSizeBytes: 524288

# Cell execution timeout. A value of 0 indicates no Studio-specific
 timeout is applied. Instead,
# use the DSE server timeouts that are configured in dse.yaml.
# Unit: milliseconds
executionTimeoutMs: 0
```

```
# Options to configure the Studio web server.
server:
 httpPort: 9091
 # WARNING: Changing the setting from the default (localhost) can pose a
 security risk as other
 # users on external machines can gain access to notebooks and the DSE
 clusters those
 # notebooks are connected to. Studio is designed to be used as a desktop
 application.
 # Distributed deployment introduces potential security risks.
 # See http://docs.datastax.com/en/latest-studio/studio/reference/
configuration.html for documentation.
 httpBindAddress: localhost

# Studio logging options.
logging:
 fileName: studio.log
 maxLogFileSize: 250 MB
 maxFiles: 10
 directory: ./logs
 # Spark SQL log level
 # 0: Disable all logging
 # 1: Log severe error events that cause the driver to stop
 # 2: Log errors that may allow driver to continue
 # 3: Log events that might results in an error
 # 4: Log general driver progress information
 # 5: Log detailed driver debug information
 # 6: Log all driver activity
 sparkSQLLogLevel: 0

# The path to the local file system where user data is stored.
userData:
  # Defaults to .datastax_studio folder in your home directory, such as
 ~/.datastax_studio
  # Set to a non-null value to override.
  baseDirectory: null

  # Time interval between revision saves when only minor changes are
 made.
  # For example, revision cell code and settings changes. Major changes,
 such as executing a cell
  # and getting a new result, always create a revision history unless the
 result is identical
  # to the prior values.
  historySaveFrequencyInSeconds: 300
  # Directory to store notebooks revisions that can be restored.
  historyDirectory: history
  # Enable Gzip compression of history files.
  compressHistoryFiles: true
  # Enable pruning of history revisions. When there are more than
 "minHistoryRevisionsToKeep"
  # history revisions, revisions older than "maxDaysOfHistoryToKeep" days
 are deleted.
```

```
  pruneRevisionHistoryEnabled: true
  # Maximum number of days to retain history revisions.
  maxDaysOfHistoryToKeep: 30
  # Minimum number of revisions to retain before pruning by date is
enforced.
  minHistoryRevisionsToKeep: 25
```

Cell options

resultSizeLimit
: Maximum number of items returned per cell execution. Additional items will be truncated. Default: 1000.

maxResultSizeBytes
: Maximum size of a cell result. If a cell result exceeds this size then the cell execution will fail. Default: 10485760.

executionTimeoutMs
: Cell execution timeout in milliseconds. A value of 0 indicates no timeout override. Uses the DSE server timeouts configured in the dse.yaml file. Default: 0.

Studio web server

httpPort
: The port on which the Studio server is running. Default: 9091

httpBindAddress
: The IP address to which the Studio server is bound. Default: 127.0.0.1.

Logging options

fileName
: Name of the log file. Default: studio.log.

maxLogFileSize
: Default: 250 MB.

maxFiles
: Maximum number of log files. Default: 10.

directory
: Path of the directory in which log files are stored. Default: ./log.

sparkSQLLogLevel
: Spark SQL log level. Default: 0

User data

The path to the local file system where user data is stored.

baseDirectory
: Defaults to .datastax_studio folder in your home directory, such as ~/.datastax_studio. Set to a non-null value to override. Default: null

historySaveFrequencyInSeconds
: Time interval between revision saves when only minor changes are made. For example, revision cell code and settings changes. Major changes, such as executing a cell and getting a new result, always create a revision history unless the result is identical to the prior values. Default: 300

historyDirectory

Directory to store notebooks revisions that can be restored. Default: ~/.datastax_studio/history

compressHistoryFiles
Enable Gzip compression of history files. Default: true

pruneRevisionHistoryEnabled
Enable pruning of history revisions. When there are more than minHistoryRevisionsToKeep history revisions, revisions older than maxDaysOfHistoryToKeep days are deleted. Default: true

maxDaysOfHistoryToKeep
Maximum number of days to retain history revisions. Default: 30

minHistoryRevisionsToKeep
Minimum number of revisions to retain before pruning by date is enforced. Default: 25

Advanced configuration options for DataStax Studio

Advanced configuration options are not explicitly declared in the `configuration.yaml` file.

DataStax recommends contacting the DataStax services team before adding advanced configuration options. For the properties in each section, the parent setting has zero spaces. Each child entry requires at least two spaces. Adhere to the YAML syntax and retain the spacing.

userData

connectionsDirectory
The directory where connections are stored. Default: connections.

snapshotSaveIntervalInSeconds
Default: 300.

entityCacheIdleTimeoutInSeconds
Default: 3600.

maxKeyspaceSessionsPerConnection
Maximum number of sessions associated with a specific keyspace to keep open at a time. Least recently used sessions are closed first. Default: 5.

eventReplayTimeoutInSeconds
Default: 600.

eventReplayBatchSize
Default: 10.

security

To make encryption of passwords unique for your installation, you can change the password in this file. Use a strong generated password. DataStax recommends following security best practices, including avoiding simple words and phrases.

encryptionPasswordFile
Default: conf/security/security.properties.

connectionManagement

idleTimeoutInSeconds

How long before an unused connection expires and is closed when it is not in use. Default: 3600 (1 hour).

Schema refresh

schemaRefreshIntervalMs
Schema refresh polling interval in milliseconds. Default: 3000 (3 seconds).

User data in DataStax Studio

Studio notebooks, connections, and other user data are stored in *`user_home_directory/`*`.datastax_studio` (default).

Warning: The `.datastax_studio` is useful only to create backups of user data. Do not change any information or copy files from the user data directories.

Specifying JVM settings for DataStax Studio

Studio enables you to specify JVM command-line options for the local Studio server. Studio runs with these defaults:

- min heap 256 MB
- max heap 4 GB
- temp dir `/tmp`

The default values for Xmms (min heap), Xmx (max heap), and temp dir are expressed as:

```
export STUDIO_JVM_ARGS="-Xms256m -Xmx4g -Djava.io.tmpdir=/tmp"
```

Adjust as appropriate for your environment.

1. Create a file named `setenv.sh` for Linux operating systems (`setenv.bat` for Windows).
2. Add the content for the JVM arguments to pass to Studio.

   The format of the content is:

   ```
   export STUDIO_JVM_ARGS="JVM_options"
   ```

   The default values for Xmms (min heap), Xmx (max heap), and tmpdir are:

   ```
   export STUDIO_JVM_ARGS="-Xms256m -Xmx4g -Djava.io.tmpdir=/tmp"
   ```

   For example, to change the maximum heap size to 8 GB:

   ```
   export STUDIO_JVM_ARGS="$JVM_OPTS -Xmx8g"
   ```

3. Save the file in the same directory as the Studio `server.sh` file.

For example: `datastax-studio/bin/setenv.sh`

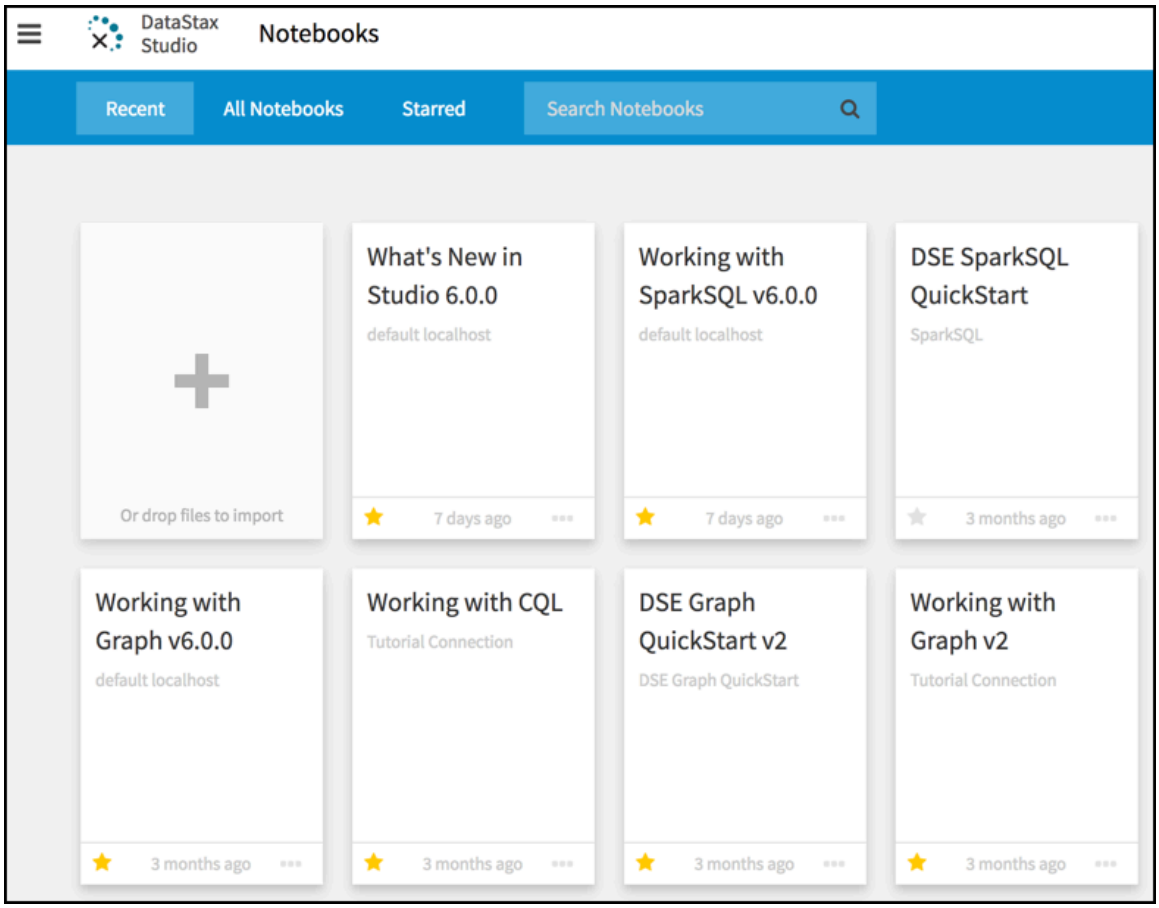
4. Restart *(page 10)* Studio to use the JVM options.

DataStax Studio 6.7 Reference

Creating a simple notebook in DataStax Studio

Steps to create a notebook that contains text and runnable code. A notebook requires a name and a connection to a DSE cluster.

1. Open the Studio server by entering `http:/`*`URI_running_DSE`*`:9091/` in your web browser.



   DataStax Studio opens to the Notebooks page.

2. Click the plus (+) to add a notebook.

   The **Create Notebook** dialog displays.

3. Enter the notebook name.
4. Select an existing connection or click **+ Add New Connection**.
5. For a new connection, enter the connection information:

Name
Name of the connection from the notebook to a DSE cluster.
Host/IP (comma delimited)
The host names or IP addresses of the DSE cluster to connect to. Default: localhost.
Username
Optional. DSE username for logging in.
Password
Optional. DSE password for logging in.
Port
IP connection port. Default: 9042.

For example, to connect to a single-node DSE cluster on the local host using the default port.

Name
My First Connection
Host/IP
`127.0.0.1`
Port
9091

6. Click **Test** to verify the connection works.
7. To configure a secure encrypted connection, select the **Use SSL** check box.

   The Truststore and Keystore fields display. See Using SSL connections in DataStax Studio *(page 13)*.

8. Click **Save**.

   A connection is created from the notebook to the DSE cluster.

9. Select **Create**.

   The notebook displays with a single empty (default) cell in **CQL** edit mode. This can be changed to Gremlin, SparkSQL, or Markdown.

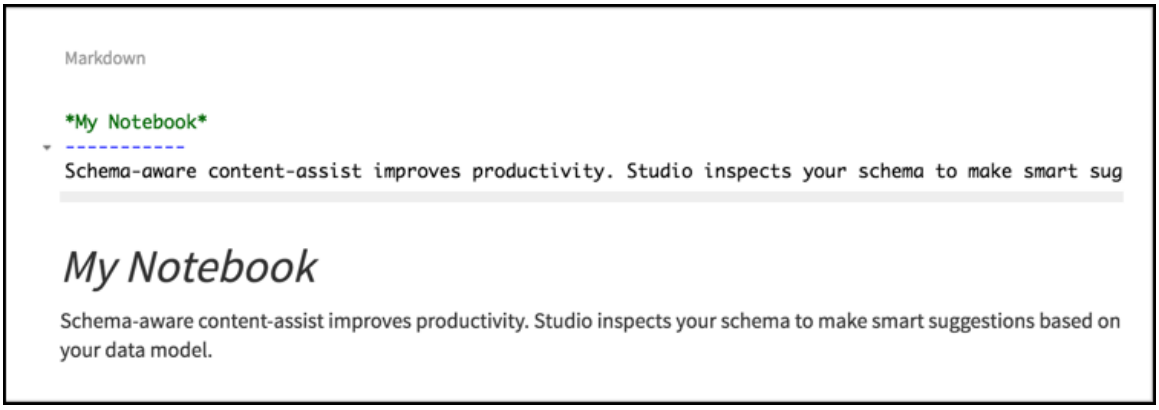
The following steps show a Markdown example:

10. In the default cell, change the drop-down to **Markdown**.
11. Add some verbiage to the default cell:

```
My Notebook
-----------
Schema-aware content-assist improves productivity. Studio inspects
 your schema to make smart suggestions based on your data model. When
```

```
you work with properties of a person, then you'll see suggestions for
name and date_of_birth.
```

12. Select **Run Cell** to render the marked down text.



13. Create another Markdown cell.

Configuring a notebook in DataStax Studio

You can change the name, connection, and graph of a notebook.

Each graph notebook is connected to a particular graph. See DSE Graph QuickStart.

A connection in Studio defines the graph and assigns a graph traversal *g* for that graph. A graph traversal is the mechanism for visiting each vertex in a graph, based on the filters defined in the graph traversal. To query DSE Graph, the graph traversal *g* must be assigned to a particular graph; Studio manages this assignment with connections.

1. To configure a notebook:
   - On the Notebook Manager page, click the ellipsis button (…) in the notebook you want to configure.
   - Open the notebook you want to configure.
2. In the menu (#), select **Configure this Notebook**.
3. You can change the name of the notebook, the connection, and the graph.
   - The notebook name must be unique. The name field prevents duplicate names.
   - Each notebook has only one connection. A connection can serve multiple notebooks.
   - Each notebook is connected to a particular graph. Multiple notebooks can be connected to the same graph.

4. Click **Save**.

Default gremlin imports in DataStax Studio

The following imports are performed by default for Gremlin cells. These imports are not configurable.

- Static imports:

```
# org.apache.tinkerpop.gremlin.process.traversal.SackFunctions.Barrier.*
# org.apache.tinkerpop.gremlin.util.TimeUtil.*
# org.apache.tinkerpop.gremlin.structure.Direction.*
# org.apache.tinkerpop.gremlin.process.traversal.Pop.*
# org.apache.tinkerpop.gremlin.process.traversal.step.TraversalOptionParent.Pick.*
# org.apache.tinkerpop.gremlin.structure.VertexProperty.Cardinality.*
# org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource.*
# org.apache.tinkerpop.gremlin.process.traversal.P.*
# org.apache.tinkerpop.gremlin.process.traversal.Order.*
# org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.*
# org.apache.tinkerpop.gremlin.structure.io.IoCore.*
# org.apache.tinkerpop.gremlin.process.traversal.Scope.*
# org.apache.tinkerpop.gremlin.structure.Column.*
# org.apache.tinkerpop.gremlin.structure.T.*
# org.apache.tinkerpop.gremlin.process.traversal.Operator.*
```

- Imports:

```
# groovy.grape.Grape
# org.apache.commons.configuration.*
# org.apache.tinkerpop.gremlin.process.traversal.strategy.verification.*
# org.apache.tinkerpop.gremlin.process.computer.bulkloading.*
# org.apache.tinkerpop.gremlin.process.computer.traversal.*
# org.apache.tinkerpop.gremlin.util.function.*
# org.apache.tinkerpop.gremlin.structure.io.*
# org.apache.tinkerpop.gremlin.process.computer.ranking.pagerank.*
# org.apache.tinkerpop.gremlin.groovy.loaders.*
# groovy.json.*
# org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.*
# org.apache.tinkerpop.gremlin.structure.*
# org.apache.tinkerpop.gremlin.process.traversal.strategy.decoration.*
# org.apache.tinkerpop.gremlin.process.traversal.engine.*
# org.apache.tinkerpop.gremlin.structure.io.gryo.*
# org.apache.tinkerpop.gremlin.process.traversal.strategy.optimization.*
# org.apache.tinkerpop.gremlin.process.traversal.step.util.event.*
```

```
# org.apache.tinkerpop.gremlin.util.*
# org.apache.tinkerpop.gremlin.structure.util.*
# org.apache.tinkerpop.gremlin.structure.io.graphml.*
# org.apache.tinkerpop.gremlin.process.computer.*
# org.apache.tinkerpop.gremlin.process.traversal.strategy.finalization.*
# org.apache.tinkerpop.gremlin.process.computer.clustering.peerpressure.*
# org.apache.tinkerpop.gremlin.structure.util.detached.*
# org.apache.tinkerpop.gremlin.structure.io.graphson.*
# org.apache.tinkerpop.gremlin.process.traversal.*
# org.apache.tinkerpop.gremlin.process.computer.bulkdumping.*
# org.apache.tinkerpop.gremlin.process.traversal.util.*
# org.apache.tinkerpop.gremlin.groovy.function.*
```

- Extra imports:

```
# com.datastax.bdp.graph.api.*
# com.datastax.bdp.graph.api.schema.*
# com.datastax.bdp.graph.api.id.*
# com.datastax.bdp.graph.api.config.*
# org.apache.cassandra.db.marshal.geometry.*
# com.datastax.bdp.graph.api.system.*
# java.time.*
```

Note: To get a list from within a Gremlin cell:

```
DseGraphImports.getInstance().getAllImports()
```

Keyboard shortcuts in DataStax Studio

Studio notebooks have keyboard shortcuts to increase your proficiency while writing code and content. Mode depends on cursor location:

- Edit mode when focus is on a cell editor.
- Command mode when focus is on the cell.

Edit mode keyboard shortcuts

##+#

Shortcut	Description
Ctrl+Shift+?	Display **Shortcut help**.
Esc	Unfocus the editor and focus the cell (switch to **Command mode**).
Shift+Enter	Save code and execute cell.

Shortcut	Description
Ctrl+Space	Schema aware content assist.
Ctrl+L	Toggle line numbers.
Ctrl+/ (Macintosh OS X: #+/)	Toggle comment.
Ctrl+H	Hide the editor and focus on the cell (switch to **Command mode**).
Ctrl+Alt+H	Add a new cell below and switch to its editor.
Ctrl+Alt+Shift+H	Add a new cell above and switch to its editor.
Ctrl+Shift+S	Toggle schema view.
Ctrl+Shift+Down (Macintosh OS X: `#+#+Down`)	Focus on the cell below, entering its **Command mode**.
Ctrl+Shift+Up (Macintosh OS X: `#+#+Up`)	Focus on the cell above, entering its **Command mode**.
Alt+Backspace (Macintosh OS X: `#+Backspace`	Delete from the cursor to the beginning of the line.
Alt+Delete (Macintosh OS X: `#+Delete`	Delete from the cursor to the end of the line.

Command mode keyboard shortcuts

Shortcut	Description
Ctrl+Shift+?	Display **Shortcut help**.
Enter	Focus on the cell's code editor (switch to **Edit mode**).
Delete (Macintosh OS X: `fn+Backspace`)	Delete the current cell and focus on next available cell.
H	Toggle editor visibility.
N	Add new cell below and switch to its editor.
Shift+N	Add new cell above and switch to its editor.
S	Toggle Schema view.
Up	Switch to the cell above.
Down	Switch to the cell below.
Shift+Up	Move current cell up.

Shortcut	Description
Shift+Down	Move current cell down.

Notebook cells in DataStax Studio

Notebook cells contain markdown text, CQL, Gremlin code, or Spark SQL. A notebook consists of a sequence of cells. A cell is a multi-line text input field.

Execute cell contents by:

- Pressing Shift + Enter with focus in the cell
- Clicking the Play button # on the right
- Selecting **Cell | Run** in the menu bar

The cell language determines the execution behavior of a cell:

- Gremlin

  Execution configurations *(page 25)* define the run behavior.

- CQL

  Execution configurations *(page 25)* define the run behavior. Custom run configurations are supported.

- Spark SQL
- Markdown

  Markdown is a superset of HTML. Use Markdown to document the computational process in a literate way, alternating descriptive text with code. You can make text italic or bold. You can build lists, and provide other structure including headings that link to other sections of the notebook. Click Play to display the formatted markdown text.

Every cell starts off being a code cell, but you can change its type by selecting a different cell language or by using Keyboard shortcuts in DataStax Studio *(page 40)*.

Notebook

Widget	Name	Description
	View Schema	Switch to Schema to view and interact with contextual views of data.
	View History	Display notebook history to view and manage notebook revisions *(page 23)*.
	Add Cell	Add a cell in the current location.

Cells

Widget	Name	Description
▶	Run Cell	Display the markdown or runs the Gremlin code and displays the results.
	Hide Code Editor	Hide the code editor.
	Disable editor validations	Toggle editor validations to off.
	Enable editor validations	Toggle editor validations to on.
	Maximize cell	Maximize the cell.
•••	More actions	Display more menu items for the cell.

Gremlin results display

Display the results returned by the last Gremlin statement executed in a cell in these ways:

Widget	View description
	Raw
	Table
	Pie chart
	Bar
	Line
	Area
	Scatter
	Graph

Widget	View description
	Settings

Schema view

Widgets for schema view:

Widget	Description
	Reset drop down selections
	Dock schema to the top
	Dock schema to the side

DataStax Studio 6.7 FAQ

Frequently asked questions

Why can I configure more than one host in a Studio connection?

Hosts initialize the Cassandra driver connection. Configuring more than one host provides redundancy and failover protection.

Which web browsers are supported for Studio?

Studio is tested on these platforms (all 64-bit) with the latest versions of the specified web browsers.

How can I view a list of my notebooks?

Use the Notebook Manager *(page 22)* to list and filter notebooks.

How do I specify a default keyspace for a cell?

You can use the keyspace menu to select the keyspace or specify the CQL USE statement. The following rules apply to USE statements in Studio:

- USE statements apply only to the current cell.
- A USE statement overrides the default keyspace setting for all statements that follow the USE statement within that cell.
- The keyspace selected in the keyspace drop down menu applies only to the current cell.
- The selected keyspace is copied to new cells created below an existing CQL cell.
- If a default keyspace is not selected, and there are no USE statements in the cell, all statements must be fully qualified to include the desired keyspace.

Does Studio support connection to SSL-enabled clusters?

Yes. See Using SSL connections in DataStax Studio *(page 13)*.

Where are my notebooks from earlier versions?

When you start Studio, all of your existing notebooks are upgraded to the new version. Notebooks are not backward compatible.

Troubleshooting DataStax Studio 6.7

Fixing problems in DataStax Studio 6.7.

See Troubleshooting DataStax Studio.