Cray® Graph Engine User Guide

(S-3010-1000)

Contents

About the Cray® Graph Engine User Guide

The Cray® Graph Engine User Guide contains information about using the Cray Graph Engine (CGE), its Command Line Interface (CLI) and Graphical User Interface (GUI) to create and use RDF databases.

NOTE: This is a draft document.

Release Information

This publication version is a revision of the version that was released on July 31, 2015 and addresses the software release `1.0UP00` of the Urika-GX system. Major changes include:

- Additional troubleshooting information
- Addition of a "Hello World" example
- Minor corrections
- Additional quick reference information

Typographic Conventions

`Monospace`	`Monospaced` text indicates program code, reserved words, library functions, command-line prompts, screen output, file names, path names, and other software constructs.
`Monospaced Bold`	**`Bold monospaced`** text indicates commands that must be entered on a command line or in response to an interactive prompt.
`Oblique` or *`Italics`*	*`Oblique`* or *`italicized`* text indicates user-supplied values in commands or sytax definitions.
Proportional Bold	**Proportional bold** text indicates a graphical user interface window or element.
\ (backslash)	A backslash at the end of a command line is the Linux® shell line continuation character; the shell parses lines joined by a backslash as though they were a single line. Do not type anything after the backslash or the continuation feature will not work correctly.
`Alt-Ctrl-f`	`Monospaced` hyphenated text typically indicates a keyboard combination.

Scope and Audience

This publication does not include in-depth information about RDF and SPARQL. The intended audience of this publication is users and system administrators. It is assumed that all the commands documented in this guide are executed via the bash shell.

About the Cray Graph Engine (CGE)

CGE is a highly optimized software application designed by high-speed processing of interconnected data. It features an advanced platform for searching very large, graph-oriented databases and querying for complex relationships between data items in the database. It provides the tools required for capturing, organizing and analyzing large sets of interconnected data. CGE enables performing real-time analytics on the largest and most complex graph problems, and features highly optimized support for inference, deep graph analysis, and pattern-based queries.

Cray Graph Engine (CGE) Features

CGE features include:

- Optimized query engine for high-speed parallel data analysis.
- Support for submitting queries, updates and creating checkpoints.
- A rich Command Line Interface (CLI).
- The CGE graphical user interface, which acts as a SPARQL 1.1 end point. This interface enables editing SPARQL queries or SPARUL updates and submitting them to the CGE database. It also accepts a set of commands that allow users to perform various tasks, such as creating a checkpoint on a database, setting Name Value Pairs (*NVPs*) to control certain aspects of data preprocessing, and query processing etc.
- SPARQL query language extension via the `INVOKE` and `PRODUCING` operators, which allow a classical graph algorithm to be passed an RDF graph and for the algorithm's results to be returned as data that is compatible with SPARQL 1.1. This enables graph algorithm library calls to be nested within a SPARQL query.
- Multi-user support.
- Compatibility with POSIX-compliant file systems.
- Database preprocessing to apply inference rules to the data, as well as to index the data.

Concepts of Operation

CGE's operational model is comprised of the follow major components:

- The graph oriented database
- Resource Description Framework (RDF)

What the Cray Graph Engine (CGE) is Not: a Relational Database

Most modern database systems use a relational representation of their data. This means that data items are stored in tables, with each row of the table holding data items that are in some way related to each other. For example, all of the data items in the same row might be associated with the same person, as shown in the following table:

Employee ID	Given Name	Family Name	Date Hired	Job Position
29650	Georgia	Smith	11/17/2001	Eng5

In practice, a relational row like the one above is contained in a table of many similar rows:

Employee ID	Given Name	Family Name	Date Hired	Job position
29650	Georgia	Smith	11/17/2001	Eng5
10926	Alex	Jones	2/5/2008	Mktng3
72219	Paul	Anderson	8/21/2005	Admin2

....

One of these fields is called the "key" and is used as the basis for looking up data from any of the other fields. In this example, “Employee ID” would probably be used as the key. The column labels, “Employee ID”, “Given name” etc. are implicit. They are not stored with the table, but with a database *schema* that is associated with the table. The schema defines each field in the relation.

The kind of information that may be associated with a scheme is shown below:

Field	Name	Datatype
0	Employee ID	Integer, min 0, max 99999
1	Given name	Character, String length < 30
2	Family name	Character, String length < 30
3	Date hired	Integer 1-12, Integer 1-31, Integer > 1985
4	Job position	Character, String length < 10

NOTE: The database schema shown above is used as an example and is entirely conceptual. There are typically many tables in a large relational database, each with its own defining scheme.

What the Cray Graph Engine (CGE) is: a Graph-Oriented Database that Uses RDF

RDF is a data representation standard that allows data from different schemas to be merged. It accomplishes this by extending the linking structure of the Web to use Uniform Resource Identifiers (URIs) in order to create triples to name a subject, an object, and the relationship or predicate between the two.

Resource Description Framework Triples

An RDF triple contains three components:

- the subject, which is an RDF URI reference or a blank node
- the predicate, which is an RDF URI reference
- the object, which is an RDF URI reference, a literal or a blank node

Hence, data items in RDF are always represented as a trio of character strings, referred to as the “*subject*”, “*predicate*” and “*object*” fields. Because they were originally intended to be unique across the Internet, components of RDF triples use the generic URI / IRI syntax (RFCs 3986 and 3987).

A triple holding the same kind of information shown in the previous relational example might look like the following:

```
<http://cray.com/example/employeeID#29650>              (subject)
<http://cray.com/example/hasGivenName>                   (predicate)
"Georgia"^^<http://www.w3.org/2001/XMLSchema#string>    (object)
```

NOTE: The above three statements should be entered on a single line and have been shown in separate lines in this document due to lack of space. Furthermore, the text: (subject), (predicate) and (object) in the above lines are shown in this document for clarity and are not part of an actual triple.

Note that the given name data item “Georgia” is expressed as an RDF literal: the value coupled with a URL-like string identifying its data type. RDF triples are intended to be self-identifying in two ways, both of which can be seen in this example:

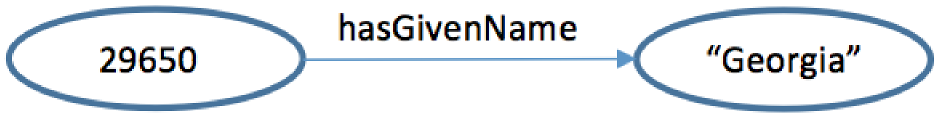
1. The literal’s data type is attached to it.
2. The predicate identifies the class of data that the object belongs to, information that in the case of relational data, is implicit in the schema and the data item’s position in the tuple. For RDF triples, there is no schema. That type of identifying information is explicit, in the predicate of the triple.

As is illustrated below, any subject-predicate-object triple can also be viewed as a source vertex-edge-sink vertex component of a directed graph:

```
<http://...ID#29650>  <http://.../hasGivenName>  "Georgia"^^<http://
www....#string>
```

NOTE: The statements shown above should be entered on a single line and have been shown on separate lines due to lack of space.

Figure 1. RDF Triple Viewed as a Graph Component



CGE is designed to store and analyze datasets when the patterns of relationships and interconnections between data items are at least as important as the data items themselves. The SPARQL query language is convenient in the sense that it provides most of the same features as SQL for filtering, grouping, and updating database information. Unlike SQL, however, SPARQL also provides a powerful mechanism for specifying (in a query) a complex interconnection pattern to search for in the database. For indefinite-size patterns and aggregate information that can’t be expressed in SPARQL,CGE supports the capability of nesting a call to a classical graph analysis function within a SPARQL query.

Each subjectpredicateobject relationship is an RDF triple. In CGE, each element in the internal representation of the database includes a graph field, which specifies the subset of the graph that the triple belongs to. If the graph field is left blank, the triple becomes part of the default graph. Typically this default, or unnamed, graph is the main data subset.

About SPARQL

SPARQL is an RDF query language developed for semantic database queries. SPARQL queries replace the table and schema format of relational SQL queries with RDF triples and ontologies, which define predicates and relationships.

Some SPARQL features are listed below:

- The ability to explore data by querying for unknown relationships.
- Implicit `JOIN` syntax, which reduces the overhead for processing a complex query with multiple `JOIN`s to the equivalent of traversing a graph.
- The `GRAPH` keyword allows data to be queried along with its source, returning both the data that matches the query and the name of the graph that contains the data.
- Unlike the proprietary query languages used in many graph database systems, SPARQL is a standardized, non-proprietary query language.
- Tools and APIs for interacting with RDF/SPARQL systems are widely and freely available for all major programming languages and platforms

This release of the CGE software supports a subset of SPARQL 1.1. The following SPARQL 1.1 features are not implemented:

- The `SERVICE` keyword, for querying remote data.
- The `MD5`, `SHA1`, `SHA256`, `SHA384`, and `SHA512` encryption functions.
- The `UCASE` and `LCASE` functions, which return a string literal whose lexical form is the upper or lower case of the lexical form of the argument, are implemented for ASCII characters only.
- The property paths feature, which extends the predicate portion of the query, allowing more extensive search patterns without the overhead of additional `OPTIONAL` statements.

  NOTE: Although CGE does not natively support the SPARQL 1.1 property paths feature, it does support certain types of property paths. CGE's property path support is currently experimental and should be used with care. Contact Cray Support for additional information.

To learn more about SPARQL, visit *http://jena.apache.org/tutorials/sparql.html*

System Architecture Overview

CGE is designed to provide performance and scalability on large, complex, interconnected databases. Its query engine is based on a data parallelism approach, in which the software strives to keep every processor busy on a roughly equal fraction of the data. The query engine is serviced by a user interface and a command line interface. See *Cray Graph Engine (CGE) Graphical User Interface* and *Cray Graph Engine (CGE) Command Line Interface* for more information.

CGE uses the open-source Jena ARQ SPARQL parser to parse each query or update, and its parser auxiliary software translates it into a lower-level representation that can drive the query engine. Query results are written to the file system in a tab-separated-values (.tsv) format. For convenience, a pointer to the results file is returned to the user when the query completes.

Extensive logging information is also written as the query or update progresses, as an aid to troubleshooting.

Major Differences Between Urika-GD and the Cray Graph Engine (CGE)

The functionality of CGE is similar to Cray Urika-GD, with a few noticeable differences that are listed below:

- Unlike Urika-GD, CGE currently does not support a “replay” capability for updates. It is required to examine the results and logs to resolve issues encountered and repeat any updates lost between user-initiated checkpoints while using CGE. The CGE application will be terminated by an error that occurs repeatedly.
- In Urika-GD, `INSERT` operations may introduce duplicates in the database and need to be eliminated by check-pointing the database. CGE does not allow duplicates to be inserted into the database and strips them out, which conforms to the SPARQL specification.
- CGE's database creation procedure is different from that of Urika-GD. Urika-GD had a multistep procedure for database creation which included importing data files, building the database and finally, loading the database. CGE enables creating and building a database in a single step via the cge-launch command. For more information on creating a database using CGE, see *Create and Use a Database*.

RDF and SPARQL Resources

Cray recommends the following resources for learning more about RDF and SPARQL:

RDF Resources

- RDF primer at *http://www.w3.org/TR/rdf-primer*

SPARQL Resources

- "*SPARQL by Example*”, available at *http://www.cambridgesemantics.com*, is an excellent introductory tutorial written by Lee Feigenbaum of Cambridge Semantics and Eric Prud’hommeaux of W3C
- SPARQL Tutorial at *http://jena.apache.org*
- "*Learning SPARQL*", available at *http://www.learningsparql.com* by Bob DuCharme
- SPARQLer Query Validator at *http://sparql.org/query-validator.html*.
- SPARQL 1.1 query language tutorial at *http://www.w3.org/TR/sparql11-query*

Semantic Web Resources

"*Semantic Web for the Working Ontologist*", available at *http://www.workingontologist.org* by Dean Alleman and James Hendler.

Cray Graph Engine (CGE) Quick Reference

The order in which CGE operations should be performed is:

Step 1: Set up SSH keys

If the following command:

```
$ ssh localhost
```

allows re-logging into the login node without a password, then the SSH keys are set up sufficiently for using CGE. On the other hand, if the previous command fails and there are existing SSH keys that do not use pass-phrases or have the `ssh-agent` defined, then try the following:

```
$ cat ~/.ssh/id_*.pub >> ~/.ssh/authorized_keys
```

At this point, if it is possible to run the aforementioned test and to re-log in to the login node without using a password, pass-phrase, or ssh-agent, then this step can be considered to be complete. If, on the other hand, the aforementioned test fails, there are no SSH keys defined yet, the following commands can be used to set them up:

> **NOTE:** Ensure that there are no existing SSH keys because this will overwrite any existing keys. Also, do not specify a pass-phrase when running `ssh-keygen`

```
$ mkdir -p ~/.ssh
$ chmod 700 ~/.ssh
$ ssh-keygen
$ chmod 600 ~/.ssh/id_*
$
$ chmod 600 ~/.ssh/authorized_keys
```

If the existing SSH key(s) use pass-phrase(s) or the `ssh-agent`, or if a more complex SSH key configuration is required, see *Cray Graph Engine (CGE) Security Mechanisms* on page 64. This section also contains information about fine-tuning access to CGE instances.

Step 2: Start the CGE Server

The `cge-launch` command launches the CGE query engine and enables creating and building a database in a single step.

The following is an example of using the `cge-launch` command:

```
$ cge-launch -o pathtoResultsDir -d path -l logfile
```

In the preceding example:

- `-o` - Specifies the path to a directory where you want the result files produced by queries to be placed.

  > **NOTE:** This path **MUST** be a directory.

- `-d` - Specifies the path to the directory containing the data set to be loaded into the server. This directory must contain all input data files for the data set.

  NOTE: This directory **MUST** contain either or both of the following if the data set is being built for the first time with CGE:

  - `dataset.nt` - This file contains triples and must be named `dataset.nt`
  - `dataset.nq` - This file contains quads and must be named `dataset.nq`

- `-l` - Specifies a log file to capture the command output from the run. This is distinct from any explicit logging performed by the database server. If the database server is logging to `stderr`, this log file will capture that information as well. There are two special argument values for this: '`:1`' and '`:2`', which refer to `stdout` and `stderr`, respectively, so that the log can be directed to either of those.

  NOTE: If the `-l` option is specified, the `cge-launch` command runs silently, producing no output to the terminal `stdout`/`stderr`.

For more information, see *The `cge-launch` Command* and *Building a Database* on page 17.

Step 3: Execute CGE CLI Commands (Optional)

CGE CLI commands can be executed after the CGE query engine has been launched. Following is an example of using the CGE `nvp-info` CLI command:

```
$ cge-cli nvp-info
```

CGE CLI features a number of commands, which are documented in the *Cray Graph Engine (CGE) Command Line Interface* on page 24 section.

Step 3: Start up the CGE Front End Server to Connect with the CGE Server (Optional)

The CGE graphical user interface and SPARQL endpoints can be accessed once the database has been launched. This can be accomplished by launching the web server that provides the user interface on a login node of the system where CGE is running.

```
$ cge-cli --ping fe
```

The `--ping` option in the preceding example is used to verify that the database can be connected to immediately upon launch and that any failure is seen immediately. Not doing so may delay and hide failures. If the ping operation does not succeed, and it is certain that the user executing this command is the only user running CGE, and that everything else is set up correctly, the user should go back to the first step and make sure that the SSH keys are set up right.

NOTE: The system may prompt to trust the host key when the `fe` command is run for the first time.

The default URL to access the UI is http://<*`hostname`*>:3756/dataset, where *hostname* is used as an example for the web server's name. For more information, see *The fe Command* on page 32.

Alternatively, the following command can be used to have the web server continue running in the background with its logs redirected, even if disconnected from the terminal session:

```
$ nohup cge-cli fe > web-server.log 2>&1 &
```

Step 4: Access and Use the CGE Front End (Optional)

For more information, see *Access the Cray Graph Engine (CGE) Graphical User Interface* on page 41.

NOTE: The system may prompt to trust the host key when the `fe` command is run for the first time.

Shutdown the CGE Server

- Shut down the CGE server using the `shutdown` command, as shown in the following example:

  ```
  $ cge-cli shutdown
  ```

  For more information, see *The shutdown Command* on page 39.
- Shut down the CGE front end if it was started.

Use the Cray Graph Engine (CGE) for a Hello World Example

Prerequisites

Ensure that the CGE application is running.

About this task

In this example, a tiny RDF triples database and query are created in such a way that it creates a sort of "Hello World" output.

Procedure

1. Create a `.nt` file. This is the original, readable representation of the database

```
<http://cray.com/example/spaceObject> <http://cray.com/example/hasName> "World" .
<http://cray.com/example/spaceObject> <http://cray.com/example/hasName> "Home Planet" .
<http://cray.com/example/spaceObject> <http://cray.com/example/hasName> "Earth" .
<http://cray.com/example/greeting> <http://cray.com/example/text> "Hello" .
<http://cray.com/example/greeting> <http://cray.com/example/text> "Hi"  .
```

2. Store the `.nt` file in the directory that has been selected or created for it.

   NOTE: This directory must be named “`dataset.nt`” if it contains triples or “`dataset.nq`” if it contains quads.

3. Select or create another directory into which the query engine should write the results and then launch the CGE server in a terminal window.

```
$ cge-launch -I 1 -N 1 -d /dirContainingExample/example -o /dirContainingExampleOutput -l :2
```

   For more information about the `cge-launch` command, and its parameters, see the `cge-launch` man page or *The `cge-launch` Command*.

   The server will output a few pages of log messages, as it starts up and converts the database to its internal representation. When it finishes, the system will display a message similar to the following:

```
$ Serving queries on nid00057 16702
```

4. In another terminal window, launch the CGE front end:

```
$ cge-cli fe
```

   When the CGE front end has been launched, a message similar to the following will be returned:

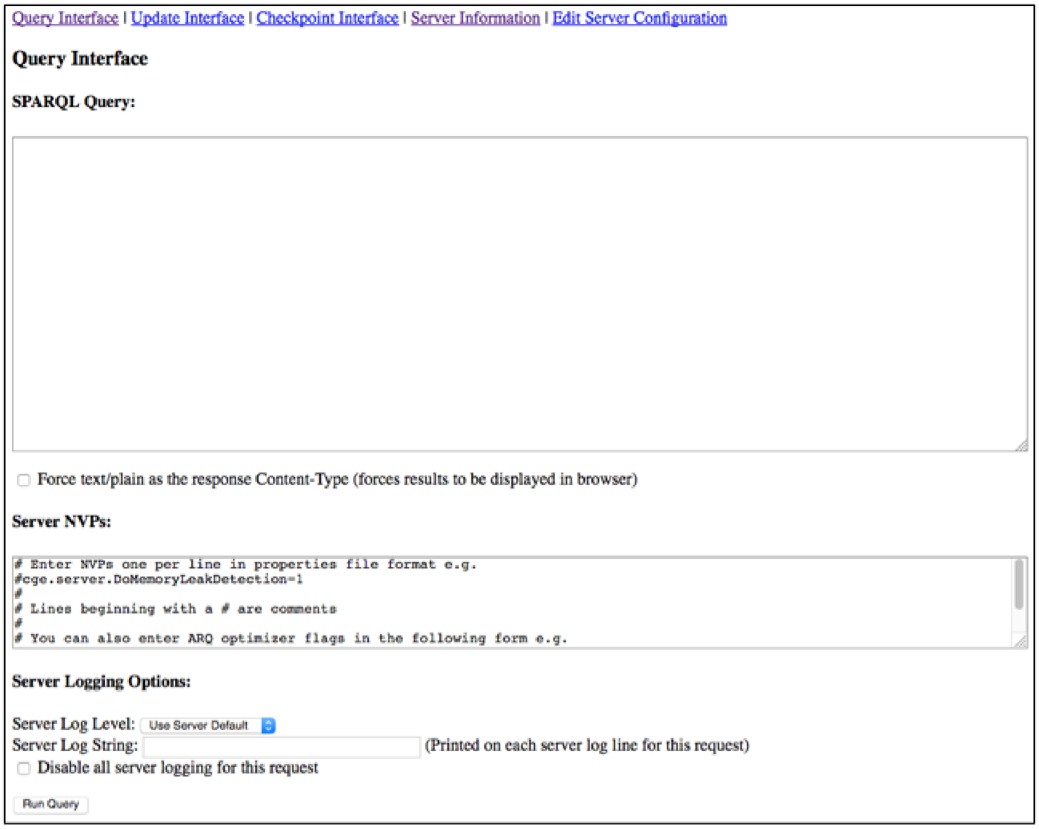
```
249 [main] INFO com.cray.cge.cli.commands.sparql.ServerCommand - CGE SPARQL Protocol Server has started and is
ready to accept HTTP requests on localhost:3756
```

   NOTE: The CGE SPARQL protocol server listens at port `3756`, which is the default port ID.

   Now the browser can be started.

5. Point the browser at the system the query engine is running on via (in the default case) port `3756`. For example, if the query engine is running on a system named *`my-machine`*, point the browser at: `http://`*`my-machine`*`:3756/dataset/`. This will connect you to the CGE front end, and windows for editing queries or updates.

Figure 2. CGE Front End



Query Interface | Update Interface | Checkpoint Interface | Server Information | Edit Server Configuration

Query Interface

SPARQL Query:

Force text/plain as the response Content-Type (forces results to be displayed in browser)

Server NVPs:

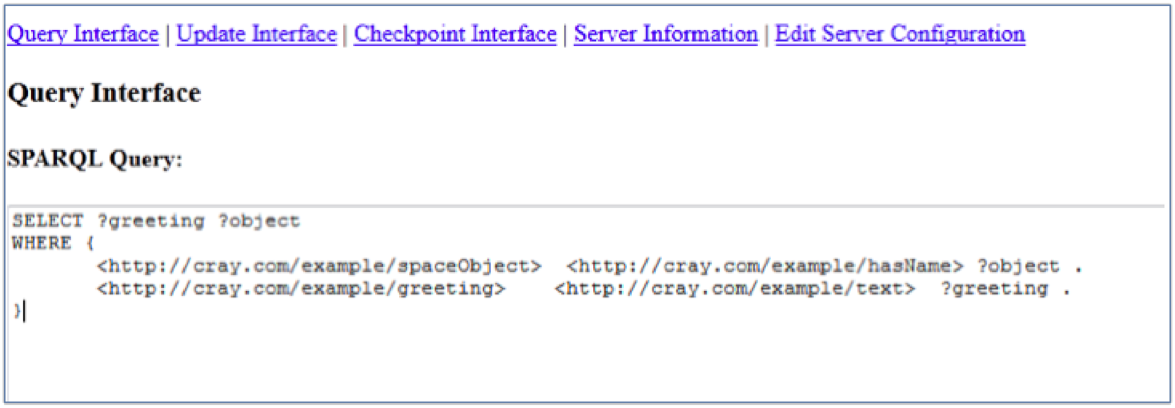
```
# Enter NVPs one per line in properties file format e.g.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
# You can also enter ARQ optimizer flags in the following form e.g.
```

Server Logging Options:

Server Log Level: Use Server Default
Server Log String: (Printed on each server log line for this request)
Disable all server logging for this request

6. Execute a query against the dataset created by typing in the query and selecting the **Run Query** button.

Figure 3. CGE Hello World Query Example



Query Interface | Update Interface | Checkpoint Interface | Server Information | Edit Server Configuration

Query Interface

SPARQL Query:

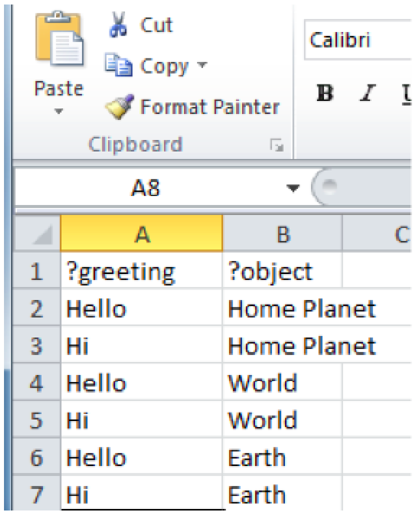
```
SELECT ?greeting ?object
WHERE {
        <http://cray.com/example/spaceObject>  <http://cray.com/example/hasName> ?object .
        <http://cray.com/example/greeting>    <http://cray.com/example/text>  ?greeting .
}
```

After the query finishes executing, the output file containing the query's results will be stored in the output directory that was specified in the `cge-launch` command. This can be verified by listing the contents of that directory and reviewing the contents of the output file, as shown in the following example:

```
$ cd /dirContainingExampleOutput
$ ls
queryResults.34818.2015-10-05T19.33.53Z000.tsv
$ cat queryResults.34818.2015-10-05T19.33.53Z000.tsv
?greeting     ?object
“Hello”       “Home Planet”
“Hi”          “Home Planet”
“Hello”       “World”
“Hi”          “World”
“Hello”       “Earth”
“Hi”          “Earth”
```

Since the Excel application can read `.tsv` files, results can also be viewed in Excel, as shown in the following figure:

Viewing CGE Output in Excel



	A	B
1	?greeting	?object
2	Hello	Home Planet
3	Hi	Home Planet
4	Hello	World
5	Hi	World
6	Hello	Earth
7	Hi	Earth

Building a Database

The Cray Graph Engine (CGE) is launched using the `cge-launch` command. When the CGE application is launched, a database directory is specified using the `-d` option of the `cge-launch` command. Initially, this directory contains RDF data in N-triples or N-quads format. When the application is first launched on a new database directory, the database is compiled and stored in an internal format in the same directory. Subsequent launches of the application using the `cge-launch` command with the same database directory will use the compiled database. The `update` command can then be used to add to or update an existing database. For more information about using the `cge-launch` and `update` commands, see *Cray Graph Engine (CGE) Command Line Interface* and the `cge-launch` and `update` man pages.

Converting Data to RDF Triples

CGE reads RDF data in N-triples or N-quads format. There are many third party tools that may be used to convert data into RDF. *D2R* is often used to extract data from an RDBMS into RDF format. The *TopBraid Composer by TopQuadrant™* can also be used to convert Excel, TSV, UML, or XML data. Conversion of data to RDF is beyond the scope of this publication.

Building the Internal Representation

Once the data has been translated into RDF, it is required to place the data in a directory where CGE can build its compiled database files. If the RDF is contained in a single file, the simplest method is to rename this file to `dataset.nt` or `dataset.nq`. A `dataset.nt` has NTriples format, whereas a `dataset.nq` file has NQuads format. On the other hand, if the RDF is found in more than one file, a file named `graph.info` will need to be created. This file contains a list of RDF files, one file per line. Each file name in `graph.info` may optionally be followed by a graph name. If a graph name is specified, the graph name is applied to any triples found in the corresponding RDF file.

Following is a sample of a `dataset.nt` file:

```
<http://www.Department14.University0.edu/GraduateStudent87>
<http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse>
<http://www.Department14.University0.edu/GraduateCourse17> .
<http://www.Department14.University0.edu/GraduateStudent87>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#TeachingAssistant> .
<http://www.Department14.University0.edu/GraduateStudent87>
<http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#teachingAssistantOf>
<http://www.Department14.University0.edu/Course6> .
<http://www.Department14.University0.edu/GraduateStudent87>
<http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse>
<http://www.Department14.University0.edu/GraduateCourse18> .
<http://www.Department14.University0.edu/GraduateStudent87>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#GraduateStudent> .
<http://www.Department14.University0.edu/GraduateStudent87>
<http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#name>
"GraduateStudent87" .
<http://www.Department14.University0.edu/GraduateStudent87>
<http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#emailAddress>
"GraduateStudent87@Department14.University0.edu" .
<http://www.Department14.University0.edu/GraduateStudent87>
<http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#undergraduateDegreeFr
om> <http://www.University843.edu <http://www.university843.edu/>> .
<http://www.Department14.University0.edu/GraduateStudent87>
<http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#advisor>
<http://www.Department14.University0.edu/AssistantProfessor6> .
```

IMPORTANT: Each predicate must appear on its own line. Some predicates are shown on multiple lines in the sample above due to lack of space.

The specification for NTriples can be found at http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/#ntriples.

Following is a sample of a `graph.info` file:

```
# example graph.info file

# filenames can be absolute
/lustre/scratch/users/jdoe/database1/dbtriples1.nt

# or they can be relative to the database directory, which is where the graph.info file resides
database2/dbtriples2.nt

# they can specify a named subgraph with a URI
/lustre/scratch/users/jdoe/database3/dbquads3.nq     <http://cray.com/namedGraphs/Graph3>
```

NOTE: Triples and quads are supported in both the `.nt` and `.nq` files. Quads in the RDF file are not affected by the optional graph name specified in the `graph.info` file. Lines containing only white space or lines beginning with the comment character (‘`#`’) are ignored. If the file is a mix of triples and quads, the triples become part of the graph specified in the `graph.info` file.

As mentioned earlier, when the application is launched via the `cge-launch` command, the `-d` parameter specifies the database directory, if it is included in the command. It is mandatory to specify this parameter, as launching CGE without it will produce an error. This directory must already exist if it has been populated with `dataset.nt`, `dataset.nq`, rules and/or a `graph.info` file. If a compiled database is not present, a database is built using the `graph.info`, `dataset.nt`, or `dataset.nq` file in that directory.

When the database has been built, the following files are saved in the database directory:

- `dbQuads`
- `string_table_chars`
- `string_table_chars.index`
- `graph.info` file is created (if not already present), which is only used to load in a database from RDF files and is not used once the database is compiled.

Once the database has been built, CGE can begin executing queries and updates. When the application is subsequently launched via the `cge-launch` command specifying the same directory, the `dbQuads` file is detected, and the compiled database is read rather than the RDF.

Memory Requirements

- **Memory Requirement for reading a database from RDF** - The amount of memory required to read a database from RDF depends on the number of triples/quads in the database, the number of unique strings in the dictionary, and the length of those strings. As a rule of thumb, however, the main memory should be 20 times the size of the RDF file(s). For example, for a 100 GiB triples file, at least 2000 GiB (20 * 100) should be used.
- **Memory Requirement for loading a compiled database** - A compiled database consists primarily of the dbQuads files, containing the compiled quads, and the `string_table_chars` files, containing the dictionary. To enable CGE to load the database and execute meaningful queries, the main memory should be 20 times the sum of the sizes of `dbQuads` and the `string_table_chars` file. For example, if `dbQuads` is 32 GiB and `string_table_chars` is 256 GiB, at least (20 * (32 + 256)) GiB of memory should be used.

About Rules Files

One way to greatly increase the knowledge contained in the database is to provide a set of inferencing rules. These rules are used during the database builds and in subsequent data updates (whether by SPARQL updates or by editing the database) to create new relationships between objects. Providing inferencing rules grants SPARQL queries access to inferred data, in addition to the raw data that was imported into the system.

Forward vs. Backward Chaining

There are two types of chaining:

- **Forward Chaining** - In forward chaining, the inferencing rules are recursively applied to the database, creating new quads and adding them to the database. If *a* implies *b* and *a* is in the database, we add *b* to the database.
- **Backward Chaining** - Rather than pre-computing quads in the database as in forward chaining, with backward chaining the queries are modified to support those rules. If ***a*** implies ***b*** and a query searches for ***b***, it is changed to search for (***a*** `UNION` ***b***).

  NOTE: CGE does not support backward chaining.

Create a Set of Inferencing Rules

Inferencing can be performed to generate additional relationships once the Cray Graph Engine (CGE) builds a database. CGE accomplishes this with a user defined rules file, which contains a set of rules specific to the data being processed. The rules file format and semantics are based on the Jena rules, documented at *http://jena.apache.org/documentation/inference/index.html#rules*. In this version of CGE there are certain limitations to these rules, which are described in *Limitations to Jena Rules Syntax* .

The rules file has the form: one or more prefixes, followed by one or more rules

```
left-hand side quad(s) -> right-hand side quad(s)
```

Comments are denoted by a # character at the beginning of a line.

The quad, or quads, on the left-hand side of the -> are the quads that the inferencer will attempt to match to infer the quad, or quads, on the right-hand side of the ->. All of the left-hand-side rules must be satisfied in order for the inference to be made. Each rule must end with a period (`.`) and a newline character, and each rule must be on its own line. The inferencer does not recognize the escape character (`\`).

A quad takes the form:

```
(subject predicate object [graph])
```

It is mandatory to specify the subject, predicate and object. The graph field is optional. If a graph is not specified, the inferencer will use the default graph and the rule will apply only to triples in that graph. The subject, predicate and object fields can be any valid form of these fields as specified by the N-Quads grammar, except as described in *Limitations to Jena Rules Syntax* in CGE. The graph field in a quad has the same valid forms as an object. If a rule contains a URI, that URI must have existed in at least one of the data files that were included in the database. Alternatively, to apply a new ontology that was not in the original data files, create a new file that contains any new objects and predicates, and add that file to the database. The fields of a quad in a rule can also be variables, or shorthand versions of strings built from a specified prefix. A variable must begin with a `?` character, followed by a valid name. A name can contain any of the following characters:

```
name := [a-zA-Z][_a-zA-Z0-9]*
```

To specify one or more prefixes at the beginning of a rules file, before any rules, use the following syntax: `@prefix prefix_name: <http://urlstring#>`

A rules file does not have to use prefixes. However they can be used to simplify quads within rules. For example, prefixes are useful for creating shorthand versions of URIs that will be used repeatedly in the rules statements.

As with rules, each prefix must end with a period (.) and a newline, and each prefix must be on its own line.

The following prefix and rule examples are from the rule set used for the LUBM data.

Inferencing a Database

When a database is built with inferencing enabled and a `rules.txt` file is found in the database directory, CGE will start applying the forward chaining rules found in that file to the triples/quads read from the RDF. The inferred quads are added to the in-memory database and stored in the compiled `dbQuads` file. If inferencing is enabled, the `rules.txt` file is also used when updating a database using SPARUL commands. As with any other quads added by the SPARUL commands, the inferred quads are added to the in-memory database but are not written to disk until the database is check-pointed.

For more information, see *About Rules Files*

> **NOTE:** Inferencing is enabled by default and may be disabled by setting the value of the `cge.serv.InferOnUpdate` control parameter to `0`. Control parameters are configuration keywords that allow controlling server configuration settings. For more information, see *Control Options*.

Examples

A prefix statement

```
@prefix ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
(?x rdf:type ub:Course) -> (?x rdf:type ub:Work) .
```

In this example the term `rdf:type` is shorthand for:

```
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>.
```

The inferencer expands the prefixed version of the string to the full string when creating the rules used during inferencing. The rule in this example says that for a given triple `?x rdf:type ub:Course` in the default graph, infer a new triple `?x is-type ub:Work` and add it to the default graph, as shown in the next example.

Inferring a new triple

Applying this rule:

```
(?x rdf:type ub:Course) -> (?x rdf:type ub:Work) .
```

to this triple in the data input:

```
<http://www.Department10.University0.edu/Course6> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Course>
```

infers (and adds) this new triple to the default graph:

```
<http://www.Department10.University0.edu/Course6> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Work>
```

A rule to establish a hierarchy of types

The following rule shows one way that ontology rules are used—to establish a hierarchy of data types.

```
(?x rdf:type ub:Faculty) -> (?x rdf:type ub:Employee) .
(?x rdf:type ub:Employee) -> (?x rdf:type ub:Person) .
```

A Faculty member is also an Employee, an Employee is also a Person, and so on. Such a rule eliminates the need to explicitly including each desired type for each such item in the database. Note that this rule did not use the graph field.

The following rule uses a variable for the graph field. This rule is excerpted from the RDFS rules file, which is based on some of the Jena rules for RDFS and OWL. The complete rules file is reproduced in Sample RDFS Rules File.

```
(?x ?a ?y ?g) (?a owl:inverseOf ?b ?g) -> (?y ?b ?x ?g) .
```

This rule is also an example of another way rules are used—to establish relationships between triples in the database. This rule states that if two predicates A and B are defined to be inverses of each other, then if the triple `(X A Y)` appears in the database, then the system can infer that the triple `(Y B X)` is also there, or should be there.

Cross-database rules

Another use of a rules file is to establish a relationship between triples in two different databases. For example, if one were extending a U.S.-based database with some additional data from France, it might streamline the process to include such rules as:

```
(<x.cray.eg.france#personne> <x.cray.eg.france#nom> ?name <x.cray.eg.frenchdb>) -> (<x.cray.eg.us#person> <x.cray.eg.us#name> ?name <x.cray.eg.usdb>) .
```

By this rule the fields in the quads are translated into their English counterparts, consistent with the data that is already in the U.S.-based database.

Sample RDFS Rules File

The following sample rules file is based on the Jena rules for RDFS and OWL. It is reproduced here courtesy of w3.org.

```
# These rules are based on the Jena rules for rdfs, plus some Jena rules
# for OWL.
#Line breaks inserted into some of these rules for formatting purposes.
#This was done for readability within this document, but is not valid syntax.

# Make a prefix for rdf:type. The IRI is defined by the SPARQL to be

# http://www.w3.org/1999/02/22-rdf-syntax-ns#type, which we can
# shorthand with rdf:type by defining a prefix for rdf:
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

# Shorthand for rdfs

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

# Shorthand for owl

@prefix owl: <http://www.w3.org/2002/07/owl#> .

# Skip this one.
# [rdf1and4: (?x ?p ?y) -> (?p rdf:type rdf:Property), (?x rdf:type
rdfs:Resource), (?y rdf:type rdfs:Resource)]
# Add rule for rdfs 2:

# [rdfs2: (?x ?p ?y), (?p rdfs:domain ?c) -> (?x rdf:type ?c)]
(?x ?p ?y ?g) (?p rdfs:domain ?c ?g) -> (?x rdf:type ?c ?g) .

# [rdfs2a: (?x rdfs:domain ?y), (?y rdfs:subClassOf ?z) -> (?x rdfs:domain ?z)]
(?y rdfs:subClassOf ?z ?g) (?x rdfs:domain ?y ?g) -
> (?x rdfs:domain ?z ?g) .

# Add rule for rdfs 3:

# [rdfs3: (?x ?p ?y), (?p rdfs:range ?c) -> (?y rdf:type ?c)]
(?x ?p ?y ?g) (?p rdfs:range ?c ?g) -> (?y rdf:type ?c ?g) .

# [rdfs3a: (?x rdfs:range ?y), (?y rdfs:subClassOf ?z) -> (?x rdfs:range ?z)]

(?y rdfs:subClassOf ?z ?g) (?x rdfs:range ?y ?g) -> (?x rdfs:range ?z ?g) .

# Add rule for rdfs 5a:

# [rdfs5a: (?a rdfs:subPropertyOf ?b), (?b rdfs:subPropertyOf ?c) ->
#       (?a rdfs:subPropertyOf ?c)]
(?a rdfs:subPropertyOf ?b ?g) (?b rdfs:subPropertyOf ?c ?g) -> (?a
rdfs:subPropertyOf ?c ?g) .

# Add rule for rdfs 6:

# [rdfs6: (?a ?p ?b), (?p rdfs:subPropertyOf ?q) -> (?a ?q ?b)]
(?a ?p ?b ?g) (?p rdfs:subPropertyOf ?q ?g) -> (?a ?q ?b ?g) .

# Skip this one.

# [rdfs7: (?a rdf:type rdfs:Class) -> (?a rdfs:subClassOf ?a)]

# Add rule for rdfs 8:

# [rdfs8: (?a rdfs:subClassOf ?b), (?b rdfs:subClassOf ?c) ->
# (?a rdfs:subClassOf ?c)]
(?a rdfs:subClassOf ?b ?g) (?b rdfs:subClassOf ?c ?g) -> (?a rdfs:subClassOf ?c ?
g) .

# Add rule for rdfs 9:

# [rdfs9: (?x rdfs:subClassOf ?y), (?a rdf:type ?x) ->
# (?a rdf:type ?y)]
# Put the quad with the most potential matches as the first quad to
# try and improve performance since since the first quads are handled
# in parallel.
(?a rdf:type ?x ?g) (?x rdfs:subClassOf ?y ?g) -> (?a rdf:type ?y ?g) .

# Add rules for inverse property from owl.

# [inverseOf1: (?P owl:inverseOf ?Q) -> (?Q owl:inverseOf ?P) ]
# [inverseOf2: (?P owl:inverseOf ?Q), (?X ?P ?Y) -> (?Y ?Q ?X) ]
# We again process the quad that most likely will have the largest number
# of potential matches first (make it first quad in rule) to prevent
# potential performance problems.
(?a owl:inverseOf ?b ?g) -> (?b owl:inverseOf
?a ?g) . (?x ?a ?y ?g) (?a owl:inverseOf ?b
?g) -> (?y ?b ?x ?g) .

# Add rule for owl transitive property.

# [transitiveProperyl: (?P rdf:type owl:TransitiveProperty),
# (?A ?P ?B), (?B ?P ?C) -> (?A ?P ?C)]
# We again process the quad that most likely will have the largest number
# of potential matches first (make it first quad in rule) to prevent
# potential performance problems.
(?a ?p ?b ?g) (?p rdf:type owl:TransitiveProperty ?g) (?b ?p ?c ?g) -> (?a ?p ?
c ?g) .

# Skip this one.

# [rdfs10: (?x rdf:type rdfs:ContainerMembershipProperty) -> (?x
rdfs:subPropertyOf rdfs:member)]
```

NOTE: Each prefix and rule must appear on its own line. Some prefixes and rules and are shown on multiple lines in the sample above due to lack of space.

Limitations to Jena Rules Syntax

This release of CGE does not support all aspects of Jena syntax and semantics for rules. Specifically:

- The `@include` construct is not supported.
- Calls to functions or built-in primitives, such as `print`, `all`, or `max` are not supported.
- The `[...]` syntax is not supported, including named rules.
- Backward chaining is not supported. Furthermore, backward syntax (`<-`) cannot be used to express forward chaining.
- If multiple premises or conclusions (quads) are specified on either side of the `->` in a single rule, each pair must be separated by a space. The use of commas as separators is not supported.
- Native `UTF-8` is not supported in rules files, however Unicode characters are supported within URIs, where they are valid syntax.

NOTE: It is important to note that turning inferencing on/off is a database level setting. Turning inferencing on can negatively impact performance. When this setting is set to `true`, the inferencer will run during the first time that the database compiles and for subsequent updates. Since the whole database is examined when inferencing occurs, turning this feature on after a period of time during which it was turned off, will still affect the data that was loaded during the period when it was turned off. In other words, if a user turns inferencing off and then adds or updates data, that data will also be inferenced once the user turns the inferencing feature on again and performs another update. The current version of CGE supports the 2004 RDF NTriples standard. The 2014 RDF 1.1 NTriples standard is not supported by CGE.

Mechanisms to Interact with the Cray Graph Engine (CGE) Database

The following mechanisms can be used to interact with the CGE database:

- CGE Graphical User Interface (GUI)
- CGE Command Line Interface (CLI)

Cray Graph Engine (CGE) Command Line Interface

The CGE CLI provides access to all the core functionality of the database via the command line. This interface is provided as part of the standard installation of CGE.

The list of available CGE CLI commands can be retrieved by executing the `cge-cli help` command without any options, as shown below:

```
$ cge-cli help
```

`cge-cli` commands are listed in the following table:

Table 1. CGE CLI Commands

Command	Description
`cge-cli checkpoint`	Requests creation of a checkpoint
`cge-cli echo`	Allows sending echo requests, which can be used to ping CGE to check if it is up and responding
`cge-cli fe`	Launches a web-based interface for accessing the server via a browser and provides SPARQL endpoints, which can be accessed via standard SPARQL APIs and tooling
`cge-cli help`	Displays help information
`cge-cli keyword-lookup`	Provides help with converting keywords between names and indexes so that the log options for using with other commands can be determined.
`cge-cli log-info`	Retrieves the current logging setup of the server
`cge-cli log-lookup`	Provides help with converting log levels between names and values so that the log options to use with other commands can be determined.
`cge-cli log-reconfigure`	Reconfigures the default logging setup of the server. The logging configuration changes persist until the server is shut down.
`cge-cli nvp-info`	Retrieves the current *NVP* setup of the server

Command	Description
`cge-cli nvp-reconfigure`	Reconfigures the default NVPs of the server. The NVP configuration changes persist until the server is shut down.
`cge-cli output-info`	Retrieves the current output directory for results from the server.
`cge-cli output-reconfigure`	Requests that the output directory for results be changed. The changes made persist until the server is shut down.
`cge-cli query`	Runs queries against the server, takes in SPARQL queries from files or from `stdin` only when no other query options are provided
`cge-cli sparql`	Runs a mixture of queries and/or updates against the server, takes in SPARQL queries/updates from files or from `stdin` only when no other input options are provided
`cge-cli update`	Runs updates against the server, takes in SPARQL updates from files or from `stdin` only when no other update options are provided
`cge-launch`	Launches the CGE Query Engine

Where more specific help for an individual command may be obtained by running the `cge-cli help` command, as shown in the following example:

```
$ cge-cli help checkpoint
```

Cray Graph Engine (CGE) Command Output

CGE CLI commands produce the following types of output:

- **Logging** - Provides diagnostic information about what a command is doing and is useful primarily for diagnosing any issues that may occur. All logging output goes to standard error.
- **Command Output** - Provides actual informational output of the command's status, such as query results, update success/failure etc. All command output is transmitted to the standard output.

As each type of output goes to a different output stream, output can easily be separated using standard shell redirection e.g.

```
$ cge-cli query example.rq > results.txt 2> query-client.log
```

The above example redirects the command output to the `results.txt` file and the logging to `query-client.log` file.

Cray Graph Engine (CGE) CLI Common Options

Certain options that are common to all commands and are provided by the CGE CLI are described in the following table:

Table 2. Common Command Line Options

Option	Argument(s)	Default Value	Example	Purpose
Communication Options				
`--db-host` `--dbhost`	Host	Localhost	`--db-host` machine	Specifies the host on which the database is running
`--db-port` `--dbport`	Port	`3750`	`--db-port 12345`	Specifies the port on which the database is running
`--i` `--identity`	Identity directory	`~/.ssh`	`-i` */my/custom/*`identity`	Specifies the path to a SSH identity directory to use for authenticating to the server. When omitted, the default `~/.ssh` identity is used.
`--trust-keys`	N/A	N/A	`--trust-keys`	When this option is set, new host keys will automatically be trusted even when running in non-interactive mode. This is useful in environments where the database port (and thus the host and port combination required to trust the key for) may frequently change. This option should only be used when connecting to trusted database servers.
Client Logging Options				
`--debug` `--verbose`	N/A	N/A	`--verbose`	Enables verbose mode which increases the amount of logging printed. All logging output goes to `stderr`, allowing it to be separated from command output, which goes to `stdout`. If the `--quiet` option is also specified, then the verbose mode takes precedence.
`--quiet`	N/A	N/A	`--quiet`	Enables quiet mode, which greatly decreases the amount of logging printed. All logging output is transmitted to `stderr`, allowing it to be separated from command output, which is transmitted to `stdout`.

Option	Argument(s)	Default Value	Example	Purpose
				If one of the verbose mode options is also specified, precedence is given to the verbose mode.
`--trace`	N/A	N/A	`--trace`	Enables extra verbose mode, which greatly increases the amount of logging printed. All logging output is transmitted to `stderr`, allowing it to be separated from the command output, which is sent to `stdout`. If the `--quiet` option is also specified, precedence is given to the verbose mode
Server Configuration Options				
`--nvp`	Name and value	N/A	`--nvp cge.DoMemoryLeakDetection 1`	Sets a *NVP* to send to the server as part of the request, usually necessary only if asked by Cray support to enable advanced options for debugging an issue.
`--log-disable`	N/A	N/A	`--log-disable`	Disables all server side logging for the request
`--log-level`	*`Log_level`*	N/A	`--log-level 16` Supported log levels include: ● `0=None` ● `1=Off` ● `2=Error` ● `4=Warn` ● `8=Info` ● `16=Debug` ● `32=Trace`	Changes the server side logging level for the request. Supported values may be obtained by using the `log-lookup` command.
`--log-string`	*`Log_string`*	N/A	`--log-level` Foo	Specifies a string that will be included in every server side log line pertaining to the request. This is useful if it is required to isolate and extract the log lines specific to a request.

Option	Argument(s)	Default Value	Example	Purpose
`--log-keyword-level`	*`Keyword_level`*	N/A	`--log-keyword-level 41 32`	Changes the server side logging level for a specific logging keyword. The database server uses a keyword-based system that enables extracting log levels specific to certain parts of the request processing or changing the log level for a specific keyword. Supported values may be obtained by using the `log-lookup` and `keyword-lookup` commands.
`--log-global-keyword`	*`Keyword`*	N/A	`--log-global-keyword 41`	Specifies that a given keyword should be included in all log lines.
Miscellaneous Options				
`-h` *`command`* --help *`command`*	N/A	N/A	`--help checkpoint`	Prints the help information for the command rather than running the command
`--batch` `--non-interactive`	N/A	N/A	`--non-interactive`	When set , this option guarantees that the script will never prompt the user for input, i.e. it will never use `stdin`. This may cause some commands to fail if they would require any user input other than the provided options. This is useful when invoking the CLI in a non-interactive context.

Cray Graph Engine (CGE) Properties File

A `cge.properties` file may be used in order to specify some options without having to explicitly state them with every command invocation. This properties file may exist in either the working directory from which the CLI is launched or under the `$HOME/.cge` directory. Where both files exist, the one in the working directory will take precedence. Enabling the verbose mode enables viewing output detailing exactly which properties file (if any) is used.

If present, the values from this file are used, unless these are specifically overridden with command line options.

Currently the following properties are supported:

Table 3. CGE Property Files

Property	Value	Equivalent Command Line Option	Description

`cge.cli.db.host`	*Host*	`--db-host`
		 `--dbhost`
			Host name of a CGE server that the CLI will connect to if the `--dbhost` option is not used.

`cge.cli.db.port`	*Port*	`--db-port`
		 `--dbport`
			Port number of a CGE server that the CLI will connect to if the `--dbport` option is not used.

`cge.cli.trust-keys`	`True` / `False`	`--trust-keys`	Eliminates the need for a first-time interactive CLI command each time you start using a server on a new TCP/IP port number combination.

NOTE: If a properties file that overrides the default value exists, the logging message will contain a warning to indicate that it has been set in the properties file. This is because an out of date properties file can interfere with correct communications with the database server without any obvious cause.

Defining Command Aliases

The properties file may also be used to define command aliases, these are essentially shortcuts to other commands. An alias is defined in the following manner:

```
$ cge.cli.alias.algebra=compile -c algebra
```

This defines a new alias algebra which simply invokes the compile command passing in the `-c` Algebra option. The CLI can then be invoked using the following command:

```
$ cge-cli algebra example.rq
```

This would compile the given query into algebra and is equivalent to running the following command:

```
$ cge-cli compile -c algebra example.rq
```

Command aliases are subject to the following restrictions:

- Aliases cannot override built-in commands.
- Aliases cannot be defined recursively, which means that an alias cannot be defined in terms of another alias.

Advanced Command Alias Definition

There are some more advanced functions that can be performed on aliases such as using positional parameters. For example, consider the following definition:

```
$ cge.cli.alias.c=compile -c $1
```

This creates an alias '`c`', which again invokes the compile command. However, this time it uses a positional parameter for the value of the `-c` option. With this definition the CLI can be invoked in the following manner:

```
$ cge-cli c rpn example.rq
```

Here the first argument after the alias is injected into the expansion of the alias so this is equivalent to running the following:

```
$ cge-cli compile -c rpn example.rq
```

NOTE: If a positional parameter receives no value, it will be passed through as-is, which will likely result in parser errors.

The `checkpoint` Command

The `checkpoint` command is used to request the creation of the checkpoint, which is a dump to disk of the current database state, optionally including a NQuads file that can be used to export the database to other tools. A checkpoint is a compiled database consisting of a `dbQuads`, `string_table_chars`, and `string_table_chars.index` file.

NOTE: This command simply takes in a path to a directory in which to create the checkpoint. This directory must exist, be writable and within the directory tree of the data directory of the server.

An example of using the `checkpoint` command is shown below:

```
$ cge-cli checkpoint /lus/scratch/user/db/cp1
```

If it is required to retrieve a NQuads file for use with other RDF and SPARQL tools that are needed for using the `–q` or `--quads` option of the `checkpoint` command, as shown below:

```
$ cge-cli checkpoint --quads /lus/scratch/user/db/cp1
```

Once the checkpoint has been created, the system will display a message saying: "`Checkpoint creation succeeded`".

Overwriting Existing Checkpoints

Note that the `checkpoint` command allows overwriting existing checkpoints. However it will do so in such a way that it guarantees that this is an atomic operation. This means that either the checkpoint is overwritten and replaced, or the previous checkpoint will continue to exist.

For more information, see the `cge-cli-checkpoint(1)` man page.

The `compile` Command

The `compile` command is used to compile SPARQL commands into the logical and/or physical plans that the database server will use to execute the command. This can be useful for understanding how the system is interpreting and optimizing a query or update.

An example of using the `compile` command is shown below:

```
$ cge-cli compile -c algebra example.rq
```

The above example would compile the SPARQL command found in the `example.rq` file into the algebra form outputting that to standard output. Multiple files can be specified in order to compile a large number of files at once.

Compilation Modes

The `-c/--compiler-mode` option is used to specify the desired compilation output type, supported values are as follows:

Table 4. Compilation Modes

Compilation Mode	Output Mode
`algebra`	The optimized SPARQL algebra for the query/update as human readable text in SPARQL Set Expression (`SSE`) format. This can be thought of as the logical plan for the query.
`raw-algebra`	The unoptimized SPARQL algebra for the query/update as human readable text in `SSE` format. This is the unoptimized logical plan for the query.
`rpn`	The physical plan for the query/update in binary form. Primarily intended for Cray developer use only.
`rpn-string`	The physical plan for the query/update in human readable text. Primarily intended for Cray developer use only.
`all`	Produces all of the above.

This option may be specified multiple times to request multiple output formats, note that if the `all` option is also specified, it would supersede any individual format requests. The `-a`/`--all` option can also be specified as a shortcut to specifying the `-c all` option.

Compilation Output

By default, compilation output goes to standard out and can be redirected to a file as desired. However, if multiple files need to be processed, or if more than one output type needs to be generated, then it is recommended to use the `-f/--files` option, which outputs a file for each input and compilation mode combination in the directory where the `cge-cli` command is being executed. The output file names are automatically generated based upon the input file name by replacing the extension with the appropriate extension for the output type:

Table 5. Compilation Output

Output Type	Output File Extension
`algebra`	`.algebra`
`raw-algebra`	`.rawalgebra`
`rpn`	`.rpn`
`rpn-string`	`.rpnstring`

For example, suppose that there is a file named `getTenRows.rq` that contains the following SPARQL query:

```
sparql query: select * {?s ?p ?o} limit 10
```

Now execute the `compile` command on the `getTenRows.rq`, as shown below:

```
$ cge-cli compile -c all getTenRows.rq --files
0 [main] INFO com.cray.cge.parser.sparql.algebra.OpAsRpnMessage  - Started Algebra to RPN message conversion
2 [main] INFO com.cray.cge.parser.sparql.algebra.OpAsRpnMessage  - Finished Algebra to RPN message conversion (3 operations)
```

The above command would create the following four files:

- `getTenRows.rawalgebra`
- `getTenRows.rawalgebra`
- `getTenRows.rpn`
- `getTenRows.rpnstring`

For more information, see the `cge-cli-compile(1)` man page.

The `echo` Command

The `echo` command is used to check that the database server is up and able to respond to requests.

As the name implies, the `echo` command simply sends some data to the database server and checks that the server echoes it back correctly.

An example of using the echo command is shown below:

```
$ cge-cli echo Test data
```

The above command sends the data "`Test data`" to the server. If the data is sent successfully, the system returns a messaging saying: "`Echoed data received and validated successfully`".

Generating Test Data

You can alternatively simply have the `echo` command generate some random data to send to the server. This can be used to test much larger requests then you would want to manually type in. For example:

```
$ cge-cli echo -g 8000
```

If the data is sent successfully, the system returns a messaging saying: "`Echoed data received and validated successfully`".

For more information, see the `cge-cli-echo(1)` man page.

The `fe` Command

The `fe` command is used to launch a web server that provides a user interface and SPARQL endpoints to CGE.

In order to stream query results over HTTP, this command must be running on a host that has access to the same file system that the database server is writing results to. Typically, this means executing the `fe` command on a login node of the system running CGE. It should be noted that since it is often required to have the user interface available for a long period, it is recommended to launch it in the background so that is resistant to terminal disconnects. See *Launch the Web Server* for more information.

For example:

```
$ nohup cge-cli fe > web-server.log 2>&1 &
```

When the CGE user interface server has started, the system returns a message indicating that the server has started and is ready to accept HTTP requests.

Once the user interface has been launched, it is possible to access the SPARQL endpoints on the machine and port displayed in the log message.

Verifying the Server Connection

It may be useful to verify that the database server is up and running when starting the web server. In this case, the `--echo` option can be used with the `fe` command, which makes the `fe` command check that the database server is up and running before launching the web server:

```
$ cge-cli fe --echo
```

Running on a Different Port

Sometimes it may be necessary to run the web server on a different port. This can be achieved by using the `--server-port` option with the `fe` command (whose default value is 3756) to supply an alternative port on which to run the web server, as shown below:

```
$ cge-cli fe --server-port 12345
```

NOTE: If an alternative port is chosen to run the web server, it is important to modify the URLs appropriately when accessing the user interface.

For more information, see the `cge-cli-fe(1)` man page.

The `keyword-lookup` Command

The `keyword-lookup` command provides the means to lookup mappings between keyword IDs and user friendly keyword names, in order to find the values that need to be passed to the log options when invoking other commands. Unlike most of the other commands, the `keyword-lookup` command does not actually contact the database.

For example, use the following command to lookup a specific keyword ID:

```
$ cge-cli log-lookup 28
```

Alternatively, a keyword ID can be looked up based upon the keyword name using the `keyword-lookup` command, as shown in the following example:

```
$ cge-cli keyword-lookup QRY
```

The `keyword-lookup` command can be used without any arguments to display the full mapping of levels to names, as shown below:

```
$ cge-cli keyword-lookup
```

For more information, see the `cge-cli-keyword-lookup(1)` man page.

The `log-info` Command

The `log-info` command retrieves information about the server's default logging configuration.

NOTE: The information returned by the `log-info` command does not necessarily reflect the logging settings for individual requests since all commands may use the *Cray Graph Engine (CGE) Command Line Interface* to change the log configuration for specific requests.

An example of using the `log-info` command is shown below:

```
$ cge-cli log-info
```

Example output of the above command is shown below:

```
$ cge-cli log-info
0 [main] INFO com.cray.cge.cli.commands.AbstractSimpleCgeCommand  - Making
request...
Server Log Configuration:
Version 1 - Printing Enabled - Default Level Info (8) - Keyword Levels Set {0-42}
```

In the above example, we can see that the server is configured with the default settings, as indicated by the text: `Default Level Info (8)`. However, in other cases we might see different settings, as shown in the following example output:

```
$ cge-cli log-info
0 [main] INFO com.cray.cge.cli.commands.AbstractSimpleCgeCommand  - Making
request...
Server Log Configuration:
Version 1 - Printing Enabled - Default Level Warn (4) - Keyword Levels Set {0-42}
Keyword TCP (Index 41) = Debug (16)
```

In the second example above, we can see that the default level has been turned down to `Warn`, but the TCP keyword is turned up to `Debug`. The server's default log configuration can be used via the *The log-reconfigure Command* command if needed. For more information, see `cge-cli-log-info(1)` and `cge-cli-log-reconfigure(1)` man pages.

The `log-lookup` Command

The `log-lookup` command provides the means to lookup mappings between log level values and user friendly log level names in order to find the values that need to be passed to the log options, when invoking other commands. It does so without contacting the database.

An example of using the `log-lookup` command for looking up the log level that has a value of 16 is shown below:

```
$ cge-cli log-lookup 16
```

Look up a level based on the name, as shown below:

```
$ cge-cli log-lookup Warn
```

Alternatively, use the `log-lookup` command without any arguments to retrieve the full mapping of levels to names, as shown in the following example:

```
$ cge-cli log-lookup
```

For more information, see the `cge-cli-log-lookup(1)` man page.

The `log-reconfigure` Command

The `log-reconfigure` command changes the default logging configuration of the server.

> **NOTE:** The information returned by the `log-info` command does not necessarily reflect the logging settings for individual requests since all commands may use the *Cray Graph Engine (CGE) Command Line Interface* to change the log configuration for specific requests.

For example:

```
$ cge-cli log-reconfigure --log-level 16
```

The system will display a message if an incorrect value is speficied for the log-level.

Upon successful execution of this command, the system returns the message: "`Received success response`".

> **NOTE:** It is recommended to verify that the log configuration changes have been implemented by using the `log-info` command. It may also be helpful to use *The log-lookup Command* and *The keyword-lookup Command* commands to determine the values that need to be passed the options, in order to configure logging settings as desired.

> **WARNING:** Do not set the server log levels to `DEBUG` or `TRACE`, especially, if the Cray Graph Engine (CGE) server is running with a large number of images.

For more information, see the `cge-cli-log-reconfigure(1)` man page.

The `nvp-info` Command

The `nvp-info` command retrieves information about the default *NVP* configuration of the server.

> **NOTE:** The information retrieved by `nvp-info` command does not necessarily reflect the NVP settings for individual requests since commands may change the NVP configuration for specific requests.

An example of using the `nvp-info` command is shown below:

```
$ cge-cli nvp-info
```

If the server's default NVP configuration needs to be changed, use *The nvp-reconfigure Command* command.

For more information, see the `cge-cli-nvp-info(1)` man page.

The `nvp-reconfigure` Command

The `nvp-reconfigure` command is used to change the server's default *NVP* configuration.

Upon successful execution of this command, the system returns a message saying: "`Received success response`".

> **NOTE:** Configuration changes are not necessarily reflected in the NVP settings for individual requests since commands may change the NVP configuration for specific requests. It is recommended to use *The nvp-info Command* command to verify that the changes have taken effect, as shown below:

```
$ cge-cli nvp-info
```

> **NOTE:** Most of the supported *NVP* have a defined range of acceptable values. Values specified outside of those ranges will be normalised into the range for that NVP. Unsupported NVPs are simply ignored, with a warning printed in the database logs and their values will not be stored by the server.

For more information, see the `cge-cli-nvp-reconfigure(1)` man page.

The `output-info` Command

The `output-info` command retrieves information about the current output directory of the server. This is the directory to which the server writes query results to for later retrieval.

An example of using the `output-info` command is shown below:

```
$ cge-cli output-info
```

The *The output-reconfigure Command* command can be used if it is required to change the server's output directory.

For more information, see the `cge-cli-output-info(1)` man page.

The `output-reconfigure` Command

The `output-reconfigure` command is used to change the server's output directory, to which it writes query results to for later retrieval.

An example of using the `output-reconfigure` is shown below:

```
$ cge-cli output-reconfigure /new/output/directory
```

> **NOTE:**
>
> After executing the `output-reconfigure` command, it is recommended to use the *The output-info Command* command to verify that the changes have taken effect, as shown below:
>
> ```
> $ cge-cli output-info
> ```

For more information, see the `cge-cli-output-reconfigure(1)` man page.

The `query` Command

The `query` command is used to execute queries against the running database. This command can be used to execute a single query or a sequence of queries.

Queries that need to be executed may be specified in a number of ways:

- By providing a list of files, which contain lists of files containing queries to be executed
- By providing the names of query files directly
- Via `stdin` (only if no queries are specified in other ways and the `--non-interactive` option is not used)

The supported input methods have the precedence shown in the list above. This means that if any list files are specified, those queries are executed before any queries are specified directly. This command may only be used to execute SPARQL queries. To execute updates, use the `update` command or to execute mixtures of queries and updates use the `sparql` command.

An example of using the `query` command is shown below:

```
$ cge-cli query --list queries.txt extra-query.rq
```

The above command will execute all the queries specified in the `queries.txt` file before executing the query specified in the `extra-query.rq` file. Executing queries by default produces only information about where to obtain the results and not the result itself.

An example of using the `query` command is shown below:

```
$ cge-cli query types.rq
```

Here we can see that the database returns the following results information:

```
0:28:1756:0:/lus/scratch/user/results//queryResults.22683.2014-12-09T11.58.36Z03.tsv::
```

This is a simple colon separated string with the fields shown in the following table:

Table 6. Status Descriptions

Column Index	Information
0	Status - will be `0` for successful queries

Column Index	Information
1	Result count - number of results returned
2	Result size - results size in bytes
3	Execution time - query execution time in seconds
4	Results location - path to the file containing the results
5	Error message - should be blank for successful queries

Results File Format

The file containing the results is in SPARQL Results TSV format and contains only the tabular results for the query. This means that if an `ASK`/`CONSTRUCT`/`DESCRIBE` query has been created, the results file will not contain the final results.

Printing Results

This simple format makes it easy to process with standard command line tools,. For example, the following command can be used to show the results in the console:

```
$ cge-cli query --quiet types.rq | cut -d : -f 5 | xargs cat
```

As noted earlier, the results file contains only the tabular results for the query. If results of an `ASK`/`CONSTRUCT`/`DESCRIBE` query are desired to be printed, see *Streaming Results*.

Streaming Results

As already seen, it is possible to use simple command line tools to extract and dump the query results to `stdout`. However, this only works for `SELECT` queries, and when the results can be accepted in `SPARQL Results TSV` format. If it is desired to retrieve the final results in an arbitrary format, the `--stream` option of the `query` command will need to be used. This option may only be used when executing a single query and it takes the `MIME` type of the desired results format.

```
$ cge-cli query --stream application/sparql-results+xml types.rq
```

Results are returned in SPARQL Results XML format. Supported formats include the following:

Table 7. Output Result Formats

Query Types	MIME Types	Output Format
`ASK` and `SELECT`	`application/sparql-results+xml`	SPARQL Results XML
	`application/sparql-results+json`	SPARQL Results JSON
	`text/csv`	SPARQL Results CSV
	`text/tab-separated-values`	SPARQL Results TSV
`CONSTRUCT` and `DESCRIBE`	`application/n-triples`	NTriples
	`text/turtle`	Turtle
	`application/rdf+xml`	RDF/XML
	`application/rdf+json`	RDF/JSON

Query Types	MIME Types	Output Format
	`application/ld+json`	JSON-LD

NOTE: Requesting a format that does not match the query type or is unknown will result in an error.

There are also three special values that may be passed to this option:

- Text
- JSON
- XML

When these values are specified, the CLI will automatically select an appropriate text (line-based), JSON or XML output format in which to stream the results, while taking into account the type of query being evaluated. For example providing `--stream text` might produce SPARQL results TSV for an `ASK`/`SELECT` query but produce NTriples for a `CONSTRUCT`/`DESCRIBE` query. When these special values are used, the exact output format will not be known in advance but will be guaranteed to fall into the general format given.

Making Multiple Queries

When multiple queries are executed, they are executed in the order specified (subject to the aforementioned precedence of list files over individual files) and the command will print a results header for each query.

```
$ cge-cli query types.rq list-graphs.rq ask-types.rq
```

It can be seen that a results header is retrieved for each query run.

For more information, see the `cge-cli-query(1)` man page

Common Errors Encountered Using the Cray Graph Engine (CGE) query Command

There are a number of common errors that may be encountered when using the `query` command:

- **Parser Errors** - Parser errors occur when a query is not in valid SPARQL 1.1 format. The error message details the unexpected token, along with what the parser was expecting at that stage of the parsing.
- **Unsupported Features** The system will also return an error message in cases where it is attempted to use an unsupported SPARQL 1.1 feature.
- **Communications Error** - Communication errors occur when the CLI cannot connect to the database. There are a variety of errors that can be encountered, depending on exactly what goes wrong with the communications between the CLI and the database server. If the same error is consistently returned, it is recommended to:
  - Add the `--trace` option to the command to get detailed information about the communications being attempted.
  - Review the log messages carefully.
  - Reviewing the server logs if possible to, as there are some situations which will manifest as client side communications errors that can be tracked down to the configuration of the server.

  For more information, see *Troubleshooting Common Cray Graph Engine (CGE) Issues* on page 81

Cray Graph Engine (CGE) Optimizer Configuration

On rare occasions, it may be required to change the query optimizer configuration. This can be performed by using the `--opt-off` and `--opt-on` options. Both of these options take the name of an optimizer flag to disable/ enable as desired.

The following example shows how to set the optimizer flag to `off`:

```
$ cge-cli query --opt-off optFilterPlacement types.rq
```

Executing the above statement would execute the query with the filter placement optimization disabled.

NOTE: If both the enabled and disabled flag options are specified, the flag will be considered as disabled. There are some flags whose values cannot be changed regardless of the options given.

⚠ **CAUTION:** Changing optimizer configuration can adversely affect query performance. Therefore, it is strongly recommend that the optimizer configuration be changed only when advised to do so by a Cray support engineer.

The `shutdown` Command

The `shutdown` command is used to instruct the Cray Graph Engine (CGE) server instance to shut down.

An example of using the `shutdown` command is shown below:

```
$ cge-cli shutdown
```

The `shutdown` command simply requests that the server be shutdown gracefully. If this command is executed by the user that owns the server process, the user will receive a success message indicating that the server has shut down. On the other hand, if this command is executed by a user who does not own the server process, the system will return an error message.

NOTE: If the server is in a bad state, then this command will not succeed. Standard Linux techniques for killing an application process should be used in this case.

The `sparql` Command

The `sparql` command is used to execute queries and/or updates against the database. It can be used to execute a single query/update or to execute a whole sequence of queries and/or updates.

Queries and updates to be executed may be specified in a number of ways:

- By providing list files which contain lists of query and/or update files to be executed
- By providing the names of query and/or update files directly
- Via `stdin` (only if no queries/updates are specified in other ways and the `--non-interactive` option is not used)

The supported input methods have the precedence shown in the list above. This means that if any list files are specified, queries specified in those list files will be executed before any queries specified directly.

This command may be used to execute a combination of SPARQL queries and updates. To execute queries only, the `query` command should be used, wheres to execute updates only, the `update` command should be used.

An example of using the `sparql` command is shown below:

```
$ cge-cli sparql --list commands.txt extra-command.ru
```

The above command will execute all the queries specified in the `commands.txt` file before executing the queries specified in the `extra-command.ru` file.

An example of using the `sparql` command is shown below:

```
$ cge-cli sparql list-graphs.rq create-graph.ru list-graphs.rq
```

Executing queries/updates using the `sparql` command produces the corresponding results for the command. This means that for queries it produces information about the results and for updates it produces a success/failure message as appropriate. To change the query optimizer configuration, see *Cray Graph Engine (CGE) Optimizer Configuration* .

For more information about the `sparql` command, see the `cge-cli-sparql(1)` man page.

Differences Between the `sparql` and `query` Commands

The major differences between the `sparql` and *The query Command* commands include:

- The `sparql` command can run a mixture of queries and updates, whereas the `query` command can run queries only.
- The `query` command can stream results directly using the `--stream` option.

The `update` Command

The `update` command is used to execute updates on a database. This command can be used to execute a single update or a sequence of updates. It supports all the common options described in *Cray Graph Engine (CGE) Command Line Interface*.

Updates to be executed may be specified in a number of ways:

- By providing list files, which contain lists of update files to be run.
- By providing the names of update files directly
- Via `stdin` (only if no updates are specified in other ways and the `--non-interactive` option is not used)

The supported input methods have the precedence shown in the list above. This means that if any list files are specified,updates contained within those files will be executed before any updates specified directly.

This command may only be used to execute SPARQL updates. If it is required to executed queries, use the `query` command. To execute a combination of queries and updates, use the `sparql` command

An example of using the `update` command is shown below:

```
$ cge-cli update --list updates.txt extra-update.ru
```

The above statement will execute all the queries specified in `updates.txt` file before executing the query specified in the `extra-update.ru` file. Executing an update returns a message indicating whether the update succeeded or failed.

```
$ cge-cli update create-graph.ru
```

Running Multiple Updates

If multiple updates need to be executed, they will be executed in the order specified (subject to the aforementioned precedence of list files over individual files) and the command will print a success or failure message for each update.

For example:

```
$ cge-cli update create-graph.ru drop-graph.ru
```

For more information, see the `cge-cli-update(1)` man page.

Cray Graph Engine (CGE) Graphical User Interface

CGE provides a simple interface for access via the browser and also provides SPARQL 1.1 protocol compliant endpoints. The CGE user interface enables you to perform a number of tasks, including:

- Executing queries
- Executing updates
- Creating checkpoints on a database
- Using advanced options for viewing and editing server configurations, as well as for performing server *NVP* and logging configuration changes.

Access the Cray Graph Engine (CGE) Graphical User Interface

To access the CGE user interface, point the browser at: http://*machine*:3756/dataset/, where *machine* is the host name of the machine where the web server is hosted. Multiple instances of CGE can be launched on the same node at different ports.

IMPORTANT: The firewall configuration of the host machine must allow for port `3756` to be accessed externally or this will not work unless the browser is running on the same host. If the site's firewall configuration does not permit this, SSH port forwarding can be used to forward the remote port to the local machine, as shown in the following example:

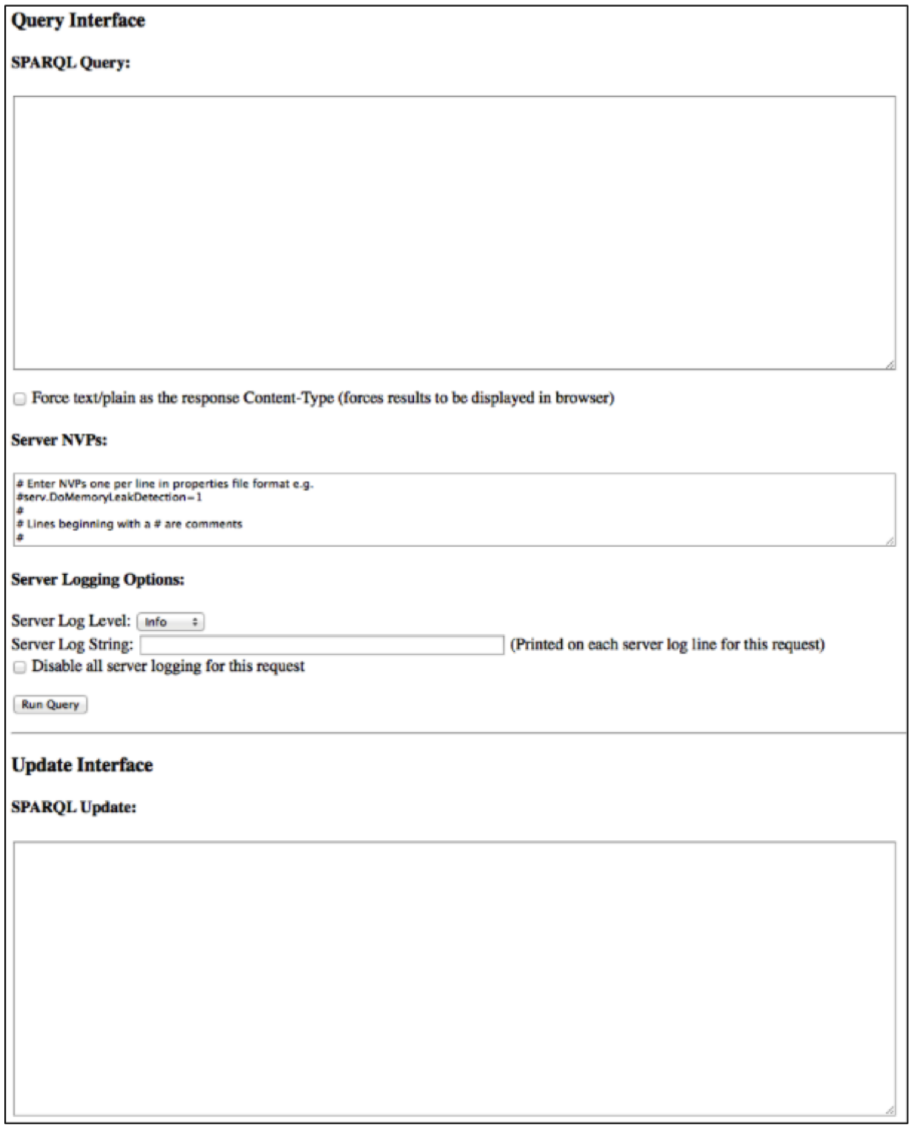
```
$ ssh machine -L 3756:hostname:3756
```

In the above example, *`machine`* is the machine running CGE’s web server. The first `3756` is the local host port to connect to, whereas *`hostname`*`:3756` is the remote reference.

The CGE UI can then be accessed by pointing the browser at: http://localhost:3756/dataset/.

Upon successfully accessing the CGE user interface the following screen will be displayed:

Figure 4. Cray Graph Engine User Interface



Query Interface

SPARQL Query:

Force text/plain as the response Content-Type (forces results to be displayed in browser)

Server NVPs:

```
# Enter NVPs one per line in properties file format e.g.
#serv.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

Server Logging Options:

Server Log Level: Info
Server Log String: (Printed on each server log line for this request)
Disable all server logging for this request

Update Interface

SPARQL Update:

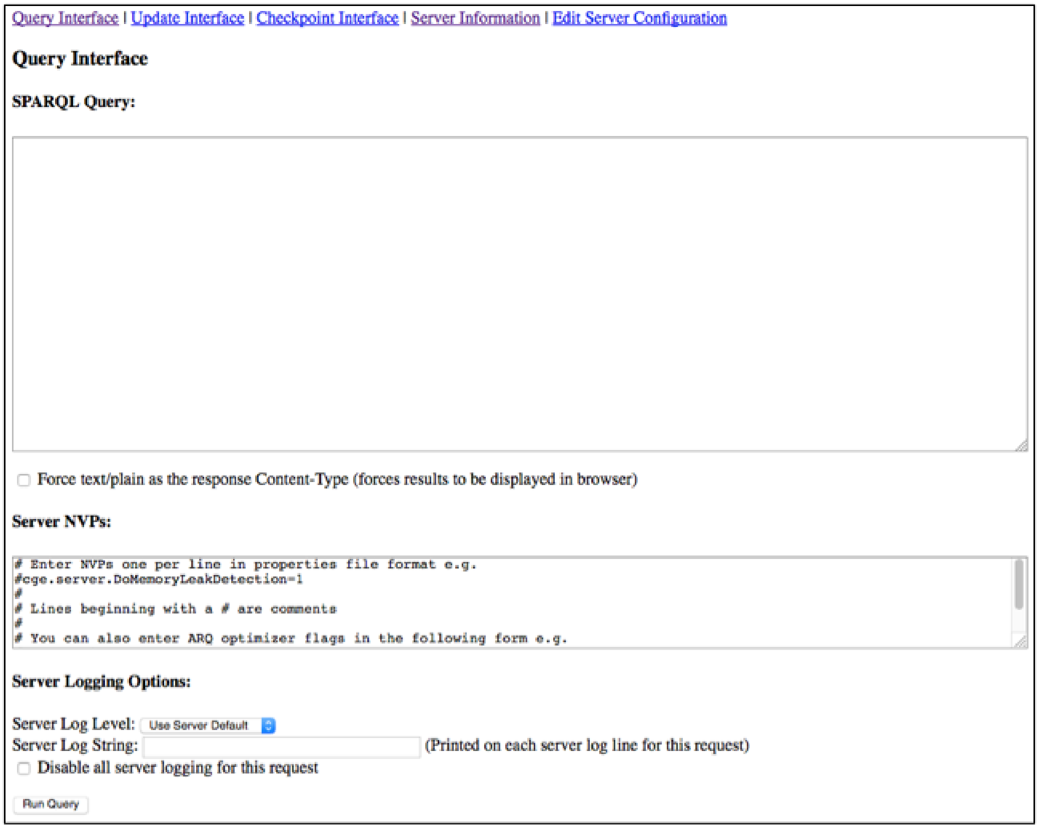
Queries and/or updates can be executed directly from this page. Additional tasks can be performed using the links at the top of this interface.

Execute SPARQL Queries

About this task

The Cray Graph Engine (CGE) **Query Interface** allows executing SPARQL queries on an loaded RDF database running within CGE.

Figure 5. Query Interface



Query Interface | Update Interface | Checkpoint Interface | Server Information | Edit Server Configuration

Query Interface

SPARQL Query:

☐ Force text/plain as the response Content-Type (forces results to be displayed in browser)

Server NVPs:

```
# Enter NVPs one per line in properties file format e.g.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
# You can also enter ARQ optimizer flags in the following form e.g.
```

Server Logging Options:

Server Log Level: Use Server Default
Server Log String: (Printed on each server log line for this request)
☐ Disable all server logging for this request

Run Query

Procedure

1. To access CGE's **Query Interface**, point the browser at http://machine:3756/dataset/query, or select the **Query Interface** link from the list of links at the top of the UI.

2. To execute a SPARQL query, enter it in the **SPARQL Query** field. The check box under the **SPARQL Query** field can be selected to specify that the server should return the query results with a `Content-Type` header value of text/plain. This will force the browser to display the results in the browser, as many browsers will download the results rather than display them by default.

3. To execute a query, simply select the **Run Query** button, which will submit the query to the server and deliver the results to the browser. The user interface uses standard HTTP content negotiation to determine the format in which to return the query results. Most browsers receive results in an `XML/JSON` format. The results of the query are written to a results file on the Lustre file system. The name of the results file is displayed on the CGE user interface. Permissions of the results file may not be correct if the user running the query and the CGE process owner are not the same.

The rest of the options seen in this interface are described in *Cray Graph Engine (CGE) Advanced Options*

Execute SPARQL Updates

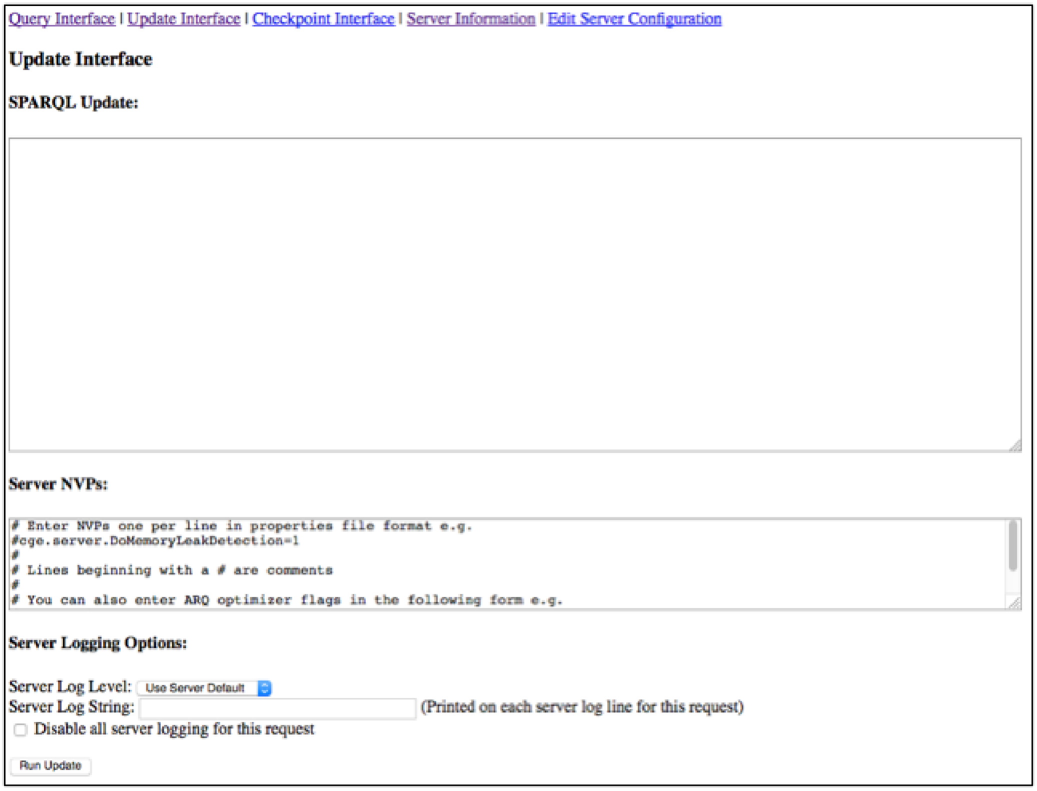
About this task

The Cray Graph Engine (CGE) **Update Interface** allows you to execute SPARQL updates on a database. SPARQL update is a language extension to SPARQL 1.1 that makes it possible to make updates to an active RDF database, using SPARQL query syntax. You can use CGE **Update Interface** interface to perform a number of tasks, including updating the default database to add or remove RDF triples and quads, copying or moving the contents of one database to another, and performing multiple update operations in a single action.

Procedure

1. To access CGE's **Update Interface** interface, point your browser at http://machine:3756/dataset/update or select the Update Interface link from the top on the main landing page of the CGE user interface.

   Figure 6. CGE Update Interface



Query Interface | Update Interface | Checkpoint Interface | Server Information | Edit Server Configuration

Update Interface

SPARQL Update:

Server NVPs:

```
# Enter NVPs one per line in properties file format e.g.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
# You can also enter ARQ optimizer flags in the following form e.g.
```

Server Logging Options:

Server Log Level: Use Server Default
Server Log String: (Printed on each server log line for this request)
☐ Disable all server logging for this request

2. To execute a SPARQL update, enter the update statement into the **SPARQL Update** field.

3. Select the **Run Update** button to submit the update for processing. Once the system has finished executing the update, it will send either a success/failure message as appropriate.

   The rest of the options seen on this screen are described in *Cray Graph Engine (CGE) Advanced Options*

Create a Checkpoint

About this task

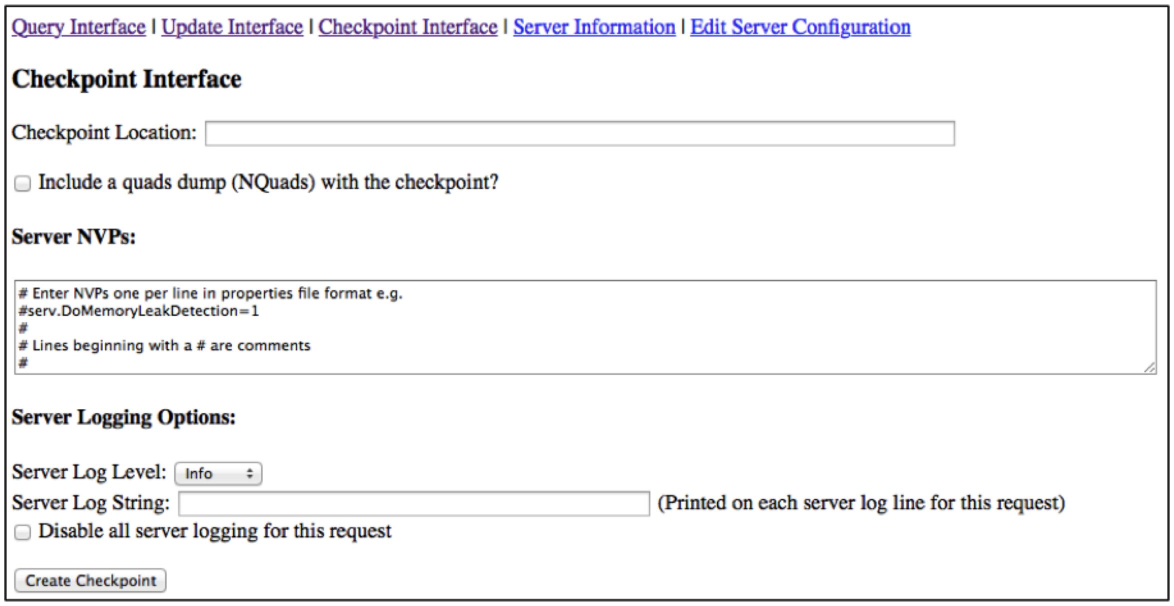
When a database is started for the first time its initial state is considered to be a checkpoint. When a change is made to the database, the state of the database can be preserved by creating a checkpoint. This preserves a copy of the previous in-memory database. Creating a checkpoint creates a persistent record of the database state, which is written to the database directory in a file named `export_dataset.nq`.

> **NOTE:** Checkpoints can only be created on running databases. If there are any queries or updates executing, it important to ensure that they finish executing before a checkpoint is created, otherwise the state of the database in the checkpoint may not contain the desired updates to it.

Procedure

1. To access the Cray Graph Engine (CGE) **Checkpoint Interface**, point the browser at http://machine:3756/dataset/checkpoint, where machine is the machine running CGE's web server. This brings up the **Checkpoint Interface**, as shown below:

   Figure 7. Creating a Checkpoint



Query Interface | Update Interface | Checkpoint Interface | Server Information | Edit Server Configuration

Checkpoint Interface

Checkpoint Location:

☐ Include a quads dump (NQuads) with the checkpoint?

Server NVPs:

```
# Enter NVPs one per line in properties file format e.g.
#serv.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

Server Logging Options:

Server Log Level: Info

Server Log String: (Printed on each server log line for this request)

☐ Disable all server logging for this request

Create Checkpoint

2. Specify a location for the checkpoint in the **Checkpoint Location** field. This is the directory where the checkpoint will be saved. The server will generate an error if this directory does not exist or is unwritable.

   > **NOTE:** This directory must already exist and must be within the tree of the database currently running. This prevents the checkpoint from accidentally overwriting other databases.

The check box under the **Checkpoint Location** field can optionally be selected to request that the database state be exported with the checkpoint. This is useful if it is required to import/export the current database state for use with other RDF-aware tools.

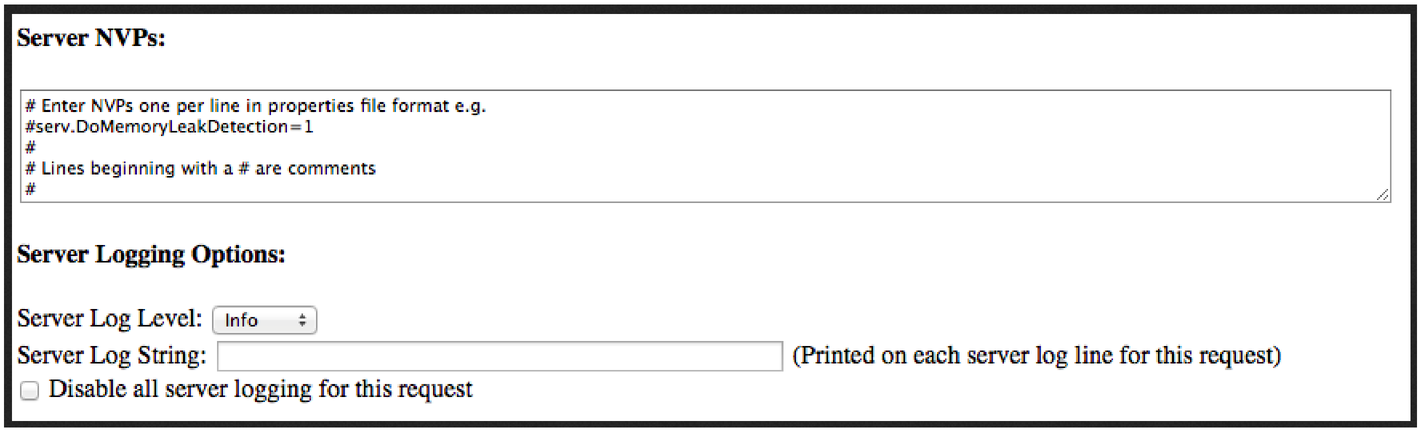
3. Select the **Create Checkpoint** button to create the checkpoint. This will return a success/failure message as appropriate, as shown in the following example output:

```
Checkpoint created at /lus/scratch/cge/datasets/lubm/0/temp
```

Cray Graph Engine (CGE) Advanced Options

CGE provides a number of advanced options that can be used to change the behaviour of the database server for a specific request. Some of these options impact the server, whereas others impact individual requests. The user interface for configuring advanced options is shown in the following figure:

Figure 8. Server Name Value Pairs



Server NVPs:

```
# Enter NVPs one per line in properties file format e.g.
#serv.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

Server Logging Options:

Server Log Level: Info

Server Log String: (Printed on each server log line for this request)

☐ Disable all server logging for this request

NOTE: Options provided in this section of the user interface are relevant only for the processing of the request under consideration and should be updated for each individual request. If it is desired to change the options for the database server as a whole, it will be required to use the interface described in the *Editing Server Configuration.*

Server NVPs (Name Value Pairs)

In the **Server NVPs** section, *NVPs* can be specified to pass to the database server. These can be used to control behaviour or enable additional debugging information.

IMPORTANT: In most cases, it will not be required to enter anything in the **Server NVPs** field, unless specifically instructed to do so by a Cray representative for gathering information to aid in diagnosing encountered issues.

Server Logging Configuration

The **Server Logging Options** section provides options that allow configuring the amount of logging the database server will produce in the server side logs during the processing of a request. The desired logging level (i.e. log verbosity) can be selected from the **Server Log Level** drop down, which is followed by the **Server Log String field**, in which a log string can be entered. The log string can be up to `128` characters and will be included on each log line pertaining to the request. This is often useful for extracting all the log lines pertaining to a specific request.

Messages of types `INFO`, `WARNING`, and `ERROR` can be logged in the system, `INFO` being the default log level.

This interface also provides the option to disable logging for the request entirely, though it is generally recommended to avoid this option as it makes it difficult to monitor the status of the server while it processes queries.

View Server Configurations

About this task

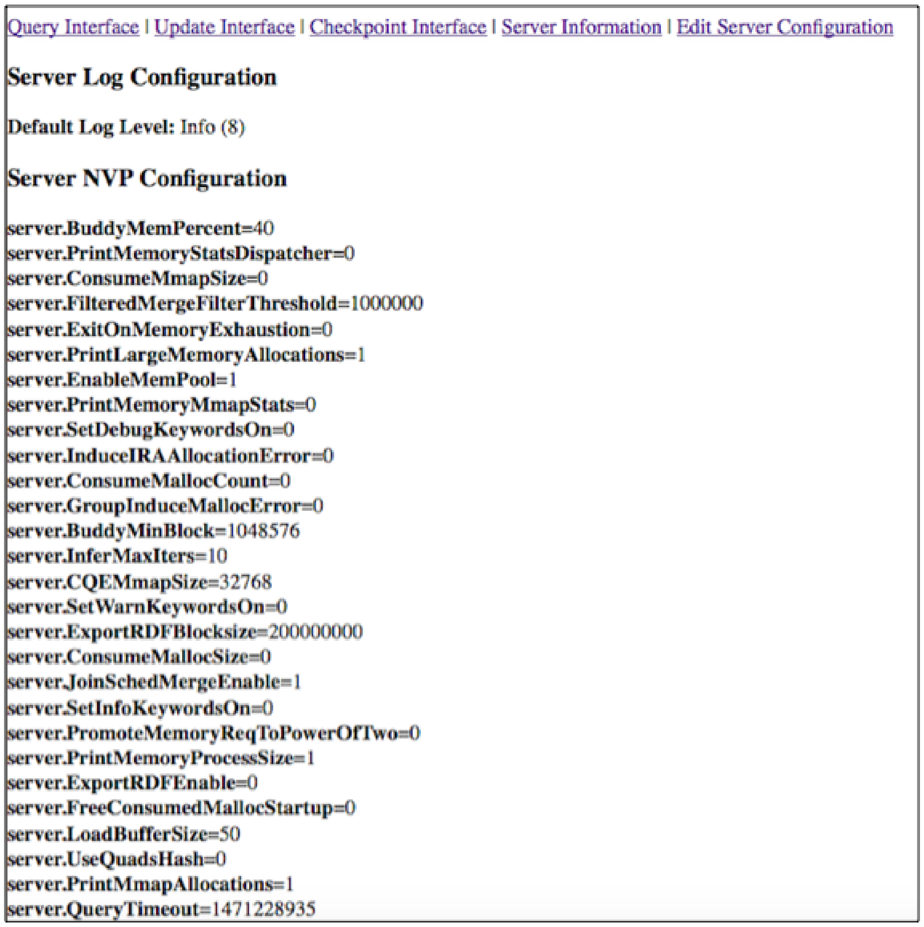
The **Server Information** interface allows you to view all the server configuration settings defined in the system.

> **IMPORTANT:** In most cases you will never need to change server configuration settings, unless specifically instructed to do so by a Cray representative in order to gather information for diagnosing issues you may be experiencing.

Procedure

To access the **Server Information** interface, point your browser at http://`machine`:3756/dataset/info, where `machine` is the machine running the Cray Gray Engine's (CGE's) web server. This displays a screen similar to the following:

Figure 9. Server Configurations


Query Interface | Update Interface | Checkpoint Interface | Server Information | Edit Server Configuration

Server Log Configuration

Default Log Level: Info (8)

Server NVP Configuration

```
server.BuddyMemPercent=40
server.PrintMemoryStatsDispatcher=0
server.ConsumeMmapSize=0
server.FilteredMergeFilterThreshold=1000000
server.ExitOnMemoryExhaustion=0
server.PrintLargeMemoryAllocations=1
server.EnableMemPool=1
server.PrintMemoryMmapStats=0
server.SetDebugKeywordsOn=0
server.InduceIRAAllocationError=0
server.ConsumeMallocCount=0
server.GroupInduceMallocError=0
server.BuddyMinBlock=1048576
server.InferMaxIters=10
server.CQEMmapSize=32768
server.SetWarnKeywordsOn=0
server.ExportRDFBlocksize=200000000
server.ConsumeMallocSize=0
server.JoinSchedMergeEnable=1
server.SetInfoKeywordsOn=0
server.PromoteMemoryReqToPowerOfTwo=0
server.PrintMemoryProcessSize=1
server.ExportRDFEnable=0
server.FreeConsumedMallocStartup=0
server.LoadBufferSize=50
server.UseQuadsHash=0
server.PrintMmapAllocations=1
server.QueryTimeout=1471228935
```

The information displayed on the **Server Information** interface includes information about the log and NVP configurations of the server, as well as the results output directory.

Edit Server Configurations

About this task

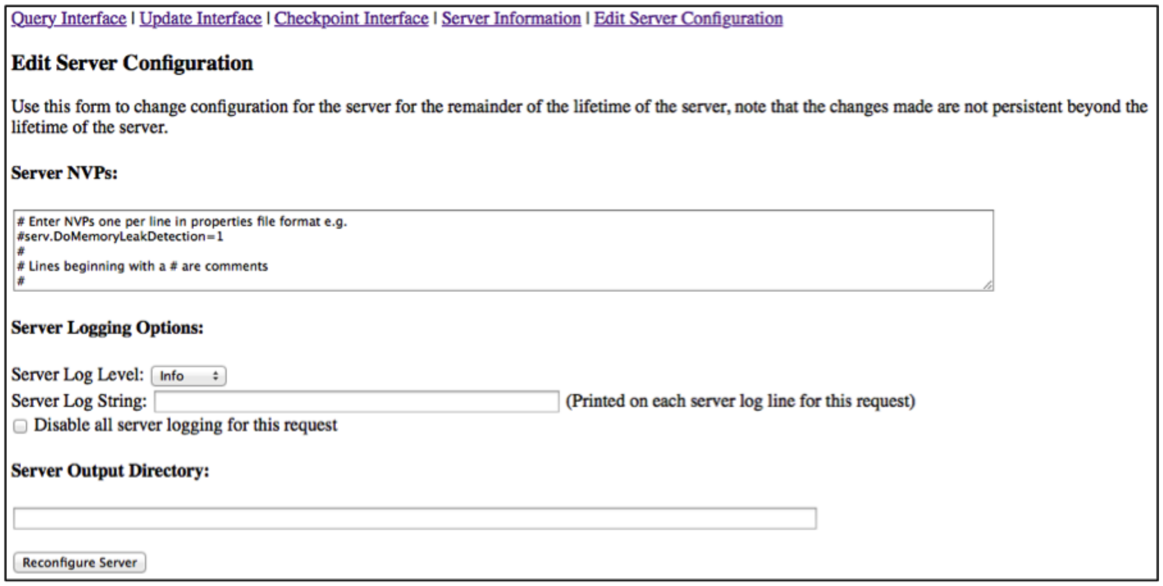
The **Edit Server Configuration** interface allows editing server configurations.

> **IMPORTANT:** In most cases you will never need to change server configuration settings, unless specifically instructed to do so by a Cray representative in order to gather information for diagnosing issues you may be experiencing.

Procedure

1. To access the Cray Graph Engine (CGE) **Edit Server Configuration** interface, point the browser at http://`machine`:3756/dataset/config, where `machine` is the machine running CGE's web server. This presents the following screen:

   Figure 10. Editing Server Configurations

   Query Interface | Update Interface | Checkpoint Interface | Server Information | Edit Server Configuration

   Edit Server Configuration

   Use this form to change configuration for the server for the remainder of the lifetime of the server, note that the changes made are not persistent beyond the lifetime of the server.

   Server NVPs:

   ```
   # Enter NVPs one per line in properties file format e.g.
   #serv.DoMemoryLeakDetection=1
   #
   # Lines beginning with a # are comments
   #
   ```

   Server Logging Options:

   Server Log Level: Info

   Server Log String: (Printed on each server log line for this request)

   ☐ Disable all server logging for this request

   Server Output Directory:

2. Select the desired serve NVP and logging options using the **Server NVPs** and **Server Logging Options** sections of the UI. In addition to the **Server NVPs** and **Server Logging Options**, this interface also contains a **Server Output Directory** field that allows changing the server output directory. This is the directory to which the database writes results, and from which the web server reads in order to deliver query results over HTTP.

   > **CAUTION:** It is best not to change server configuration settings, as doing so can adversely affect performance (especially if it is changed to point to a relatively slow file system).

3. When the required changes have been made, select the **Reconfigure Server** button to attempt the reconfiguration. Upon doing so, the system will return a response detailing the success/failures of the pieces of configuration that were to be updated, as shown in the following example output:

   ```
   Updated Server NVP Configuration successfully
   Updated Server Logging Configuration successfully
   ```

   > **NOTE:** Unlike the *Cray Graph Engine (CGE) Advanced Options* presented in the other interfaces, the values set from this interface persist for the lifetime of the server and become the new defaults.

Control Options

In most cases it will not be needed to change server configuration settings, unless specifically a Cray support represented specifically requests for doing so, in order to gather information for diagnosing issues. However, there are two such settings that you may occasionally wish to change. Name Value Pairs (NVPs) that enable you to modify these settings are listed in the following table:

Table 8. CGE NVPs

Parameter	Description	Default Value
`cge.server.QueryTimeout`	This parameter sets the maximum runtime (within the server) of a given query in seconds (wall clock time). This timeout will be checked after every operation. However, it does not interrupt operations. After the query times out, the server will terminate that query and will be immediately ready for subsequent queries. Acceptable values for this parameter range from 0 seconds (automatic termination at the start of the second operation) to `100,000` years expressed in seconds (`3153600000000`). If a negative value is entered for this field, it will be converted to `0`.	`31536000`
`cge.server.InferOnUpdate`	Causes inferencing to be enabled or disabled for a given update. Has a value of either “`0`” or “`1`”. The default value of this parameter is "`1`", which sets inferencing on for updates. A `rules.txt` file must be present for inferencing to take place. If no `rules.text` file exists, inferencing will not be performed. If updates to the database were made after inferencing was turned on, triples added previously will stay saved in the database if inferencing is turned off subsequently.	`1`

SPARQL Endpoints

Standard SPARQL tools can be used to interact with the Cray Graph Engine (CGE) by pointing them at the relevant endpoint URLs, which are shown in the following table:

Table 9. SPARQL Endpoints

Service	Endpoint URL
SPARQL Query	http://*`machine`*:3756/dataset/query
SPARQL Update	http://*`machine`*:3756/dataset/update

In the above examples, *`machine`* is used as an example for the name of the machine running CGE’s web server.

These endpoints are SPARQL 1.1 protocol compliant and provide all the standard parameters.

Supported Content Types

The SPARQL query endpoint uses standard HTTP content negotiation to determine how to return query results to the SPARQL tool, depending on the **Accept** header that the tool sends.

The following standard formats are supported by the query endpoint:

Table 10. Query Types and Supported Content Types

Query Type	Supported Content Types
`ASK` and `SELECT`	• SPARQL Results XML • SPARQL Results JSON • SPARQL Results CSV • SPARQL Results TSV
`CONSTRUCT` and `DESCRIBE`	• NTriples • Turtle • RDF/XML • RDF/JSON • JSON-LD

Standard HTTP behaviour of returning the message "`406 Not Acceptable`" will apply if the tool does not include any formats the endpoint can produce in its **Accept** header.

Custom Parameters

CGE features endpoints that provide custom parameters, which can be used to configure the same advanced options supported by the CGE user interfaces. These parameters are listed in the following table:

Table 11. Custom Parameters

Parameter	Example	Purpose
`forcePlainText`	`forcePlainText=true`	Used to force the returned Content-Type to be text/plain regardless of the actual content type being returned. This is typically only useful for browser access to the endpoints and may cause errors if used with SPARQL tools.
`nvps`	`nvps=foo%3Dbar`	Specifies the NVPs to be passed to the database and applied to the request. These must be specified in Java properties file style with one *`name`*=*`value`* pair per line
`log-level`	`log-level=16`	Specifies the log level to use for database logging of the request. This takes an integer value with values interpreted as follows: • `2 = Error` • `4 = Warn`

Parameter	Example	Purpose
		• `8 = Info` • `16 = Debug` • `32 = Trace` The `log-lookup` command can be used for translating integer values to the desired log levels.
`log-string`	`log-string=Foo`	Specifies a string to be included on every database log entry pertaining to the request. Maximum supported length is `128` characters and longer strings will be truncated accordingly.
`log-disable`	`log-disable=true`	Can be set to disable all database logging for the request

Launch the Web Server

Before using the Cray Graph Engine GUI, it is required to launch the database via the `cge-launch` command and leave the default port setting of `3750` unchanged. If an alternative port has been used, then it will be required to add the `--db-port` option to specify an alternative port. Once the database has been launched, the Cray Graph Engine (CGE) graphical user interface and/or the SPARQL endpoints may be used. This can be accomplished by launching the web server that provides the user interface on a login node of the system where CGE is running, as shown below:

```
$ cge-cli fe
```

Alternatively, you can use the following command to have the web server continue running in the background with its logs redirected, even if you disconnect from the terminal session:

```
$ nohup cge-cli fe > web-server.log 2>&1 &
```

NOTE: The web server is launched by the same script as the rest of the Command Line Interface tools, and supports many of the same standard options detailed in *Cray Graph Engine (CGE) Command Line Interface*.

Create and Use a Database

Prerequisites

If you need the Cray Graph Engine (CGE) to perform inferencing on your data, ensure that a valid `rules.txt` file exists in the directory containing your data.

About this task

The instructions listed below can be used to create a database and execute queries and/or updates on the database once it has been built.

Procedure

1. If your data is not in RDF format, convert your data to RDF using the tools mentioned in *Converting Data to RDF Triples*.

2. If your RDF data resides in a single file, save/rename that file to `dataset.nt` or `dataset.nq`. This is required because CGE accepts ONLY files in .nt or .nq formats as input. All other formats should be converted to either .nt or .nq (including .rdf. On the other hand, if you data resides in more than one file, create a `graph.info` file and add the names of your RDF file to that file. An example of the `graph.info` file is shown in *Building the Internal Representation*.

3. Build the database using the `cge-launch` command as shown below:

   ```
   $ cge-launch -o pathtoResultsDir -d path -l logfile
   ```

   In the above statement, *`pathtoResultsDir`* is used as an example for the path to the directory that will contain the results of queries and/or updates, *`path`* is used as an example for the path to the database directory and *`logfile`* is used as an example for the log file that will contain the command and server output.

   NOTE: *`pathtoResultsDir`* MUST be a directory and MUST contain either a triples or quads file. These files must be named `dataset.nt` or `dataset.nq` respectively.

   For more information, see the `cge-launch(1)` man page.

   NOTE: When the database has been built, the following files are saved in the database directory:

   - `dbQuads`
   - `string_table_chars`
   - `string_table_chars.index`

   Collectively, the aforementioned files are the disk representation of the binary version of the database which can be reloaded into CGE. When you launch the CGE application again and specify the same database directory, the `dbQuads` file will be detected and the compiled database will be read instead of the RDF. Furthermore, if the database directory contains a `rules.txt` file, CGE will perform inferencing on your data. This is because inferencing is turned on by default. It an be turned off by setting the `cge.server.InferOnUpdate` NVP parameter to `0`.

4. Execute the `fe` command to launch a web server that provides a user interface and SPARQL endpoints to CGE.

   ```
   $ nohup cge-cli fe > web-server.log 2>&1 &
   ```

   For more information about the `fe` command, see *The fe Command* on page 32.

5. To execute a query or update on your database, you can use either the CGE UI or the CGE CLI.

   a. To execute queries/updates via the CGE UI, follow the instructions listed below:

      1. Launch the CGE UI by pointing your browser at: `http://`*`machine`*`:3756/dataset/`. This brings up the CGE UI.
      2. Select the **Query Interface** or **Update Interface** to execute queries and updates correspondingly. For more information about using these interfaces, see *Execute SPARQL Queries* and *Execute SPARQL Updates*. Optionally, server configuration parameters can also be specified to control your query/update. See *Cray Graph Engine (CGE) Advanced Options* for more information.

b. To execute queries/updates via the CGE CLI, use the `query`, `update` and `sparql` commands to execute SPARQL queries, updates and/or combination of queries and updates correspondingly. For usage information, see the associated man pages.

Built-in Graph Functions

SPARQL is intrinsically designed to find explicit patterns in graphs, using the basic graph patterns called out in SPARQL specifications. Often these patterns themselves create a graph that needs to be analyzed in a way that is not easily implemented with SPARQL's basic graph patterns. One example of this in the Lehigh University Benchmark (LUBM) ontology would be to find students who take courses from their advisers, and then find the shortest path through a social network between specific pairs of those students. Another example is to use betweenness centrality to find the most "central" (i.e., connecting the most entities not otherwise connected) entities in a graph, often a social network.

To address this other type of processing, CGE's SPARQL implementation has been extended to incorporate graph-function capability. This means that the input to the graph function is a graph, not just a few scalars, such as numbers or IRIs. This capability includes both the syntax that enables calling of graph functions, and a small number of built-in graph functions (BGFs) that are callable by any CGE user.

The built-in graph functions included in this release of CGE are:

- **Betweenness Centrality**: Ranks each vertex by how frequently it is on the shortest path between vertices.
- **S-T Connectivity**: Finds the shortest path, if one exists, between two vertices in the graph.
- **S-T Set Connectivity**: Finds the shortest path, if one exists, between a set of vertices designated as sources and a set of vertices designated as targets.
- **Label Propagation**: Detects communities in networks and assigns vertices in the graph to communities.
- **BadRank**: Assigns a "badness" score to all vertices in the graph based on their nearness to known bad vertices.

Combining Graph Algorithms with SPARQL

CGE provides an infrastructure for calling graph algorithms from within SPARQL queries. A graph algorithm is called via a CGE-specific SPARQL operator named `INVOKE`.

It is useful to note the following items:

1. The `INVOKE` operator cites the name of the graph algorithm being invoked, using an URI notation that is similar to that used for representing built-in functions in SPARQL.
2. Scalar arguments can be input to the graph algorithm via a parenthesized argument list.
3. The `INVOKE` clause is always preceded by a SPARQL `CONSTRUCT` clause, whose function in this context is to build the graph that is input to the graph algorithm. CGE provides the capability of nesting a `CONSTRUCT/INVOKE` clause within a `SELECT/WHERE` clause. This enables a subquery within a SPARQL query to select or produce a subgraph, which is used as input to the graph algorithm.
4. The `INVOKE` clause is immediately followed by a `PRODUCING` clause, whose function is to bind the results of the graph algorithm to specific SPARQL variables.
5. While RDF graphs may define many different types of subjects and objects, the CGE graph algorithms treat them all as homogeneous vertices and do not distinguish between them according to type, with the exception of functions that explicitly expect some vertices to be distinguished.
6. The `CONSTRUCT-INVOKE-PRODUCING` combination needs to be nested within a `SELECT-WHERE` clause.

7. For all CGE-specific built-in graph functions, if the query writer wants to specify a non-default value for an argument, values for the preceding arguments also need to be specified, even if default values for those arguments are to be used.

Invocation of a Graph Function

Four SPARQL constructs are involved while invoking graph functions. These include:

- `CONSTRUCT`
- `INVOKE`
- `PRODUCING`
- `SELECT-WHERE`

The `CONSTRUCT` Clause

There are three main differences between a standard SPARQL `CONSTRUCT` clause and the way it is used in CGE in a `CONSTRUCT-INVOKE-PRODUCING` combination. These differences are described below:

1. As mentioned above, the `CONSTRUCT-INVOKE-PRODUCING` combination always appears nested within the `WHERE` clause of a `SELECT` query.
2. While a standard SPARQL `CONSTRUCT` query returns an RDF graph to the user, the `CONSTRUCT` clause of a `CONSTRUCT-INVOKE-PRODUCING` combination does not return anything to the user; instead the constructed graph is passed to the graph algorithm as input, and then discarded after the graph algorithm completes execution.
3. Because the output of the nested `CONSTRUCT` clause is eventually discarded, CGE relaxes some of the rules for constructing RDF graphs. In particular, since some graph algorithms expect weighted edges. CGE allows predicates to be literals inside a nested `CONSTRUCT` clause.

The `INVOKE` Clause

In CGE, graph functions are invoked using the CGE-specific `INVOKE` keyword with the `CONSTRUCT` query form. The syntax of the `INVOKE` keyword is shown below:

`INVOKE` <http://cray.com/graphAlgorithm.*`graph_function`*> (*`arguments`*)

In the above example, *`graph_function`* is the name of the graph function to be invoked and arguments is a comma-separated list of arguments to be provided to the graph function. The types and number of *`arguments`* in this list are dependent on the function being invoked.

Using the `INVOKE` Keyword

```
SELECT *
       WHERE {
          CONSTRUCT {
           ?s ?p ?o .
         } WHERE {
            ?s ?p ?o .
         }
      INVOKE <http://cray.com/graphAlgorithm.graph_function> (42,0.19,“string”)
      PRODUCING ?varX ?varY
}
```

In the above example, the `INVOKE` keyword is used to invoke a graph function named "graph_function” with three scalar arguments as well as the graph produced by the `CONSTRUCT` clause.

The `PRODUCING` Clause

In the Cray Graph Engine (CGE), the invocation of a graph function results in an intermediate result set. Ultimately, this is what enables graph functions to be composed with other SPARQL operators such as `UNION`, `ORDER BY`, or `FILTER`, as they also output an intermediate result set. The `PRODUCING` keyword can be used to bind the columns of the returned intermediate result set to SPARQL variables. The `PRODUCING` keyword accepts a list of SPARQL variable names which will be bound to the columns of the intermediate result set returned by the `INVOKE` keyword. Therefore, while using the `PRODUCING` keyword, it is required to know the following:

- How many columns will exist in the returned intermediate result set
- What set of values each column represents

The syntax of the `PRODUCING` keyword is shown below:

```
PRODUCING ?varA ?varB
```

In the above statement, `?varA and ?varB` are the variables specified by the `PRODUCING` operator to be bound to columns of the returned vectors of results.

Using the PRODUCING Clause

The community detection algorithm returns two columns of information. Information contained in these columns is described below:

- The first column contains each of the vertex IDs of the graph that was sent to the algorithm.
- The corresponding entry in the second column contains an integer that represents the identity of the community to which that vertex was assigned.

Thus the `PRODUCING` clause would specify variables that the query author chose to reflect the two vectors of data being returned, as shown in the following query snippet:

```
…
INVOKE  <http://cray.com/graphAlgorithm.community>( )
PRODUCING ?vertexID ?communityID
…
```

Inputs to the Graph Function

Three types of inputs to a graph algorithm are possible:

1. The graph itself – Each graph function expects input to come from the output of the preceding `CONSTRUCT` operator.
2. Scalar inputs – Scalar values can be passed to the graph algorithm via a parenthesized list in the `INVOKE` clause.
3. Vector inputs – Sets of values can be input to the graph algorithm by adding them to the graph that the `CONSTRUCT` operator builds. Generally these inputs are distinguished in the input graph by a triple with a type predicate and a special type object.

In the following example, the Bad Rank algorithm expects to receive a set of vertex IDs of vertices considered to be “spam” (it could represent some other undesirable attribute). Note that the `WHERE` clause associated with the `CONSTRUCT` clause includes a `VALUES` clause, that names a set of vertices that are to be considered spam by the Bad Rank algorithm. That set of vertices is added to the `CONSTRUCT` clause’s graph as a set of triples with a rdf:type predicate and the special object “`cray:spamNode`”. The scalar argument list of the `INVOKE` clause also specifies that this “`cray:spamNode`” object is to be used for identifying spam vertices. Similarly, a vector input to the graph algorithm can already be present in the database.

Using Vector Inputs for Graph Algorithm

```
PREFIX <cray: http://cray.com>
SELECT ?vertex ?ranking
{
  CONSTRUCT{
     ?sub ?pred ?obj .
     ?badNode a cray:spamNode .
    }
  WHERE {
    {
      ?sub ?pred ?obj .
    } UNION {
      VALUES ?badNode {
         <http://www.Department5.University0.edu/Course34>
          <http://www.Department6.University0.edu/GraduateCourse34>
         <http://www.Department14.University0.edu/GraduateCourse31>
         <http://www.Department5.University0.edu/Course34>
         <http://www.Department10.University0.edu/GraduateCourse25>
         <http://www.Department11.University0.edu/Course11>
         <http://www.Department13.University0.edu/GraduateStudent87>
      }
    }
  }
  INVOKE cray:graphAlgorithm.badrank (0.0001, .84, 0.01, cray:spamNode)
  PRODUCING ?vertex  ?ranking
}
ORDER BY DESC (?ranking)
LIMIT 100
```

The above example shows the invocation of the Bad Rank algorithm with a set of spam vertices present in the input graph.

Sequence of Operators

The PRODUCING operator needs to immediately follow the INVOKE operator, which in turn needs to immediately follow the WHERE clause containing the CONSTRUCT operator. The CONSTRUCT-INVOKE-PRODUCING combination should always appear as a nested subquery inside a SELECT clause's associated WHERE clause. Graph algorithms, like SELECT clauses themselves, can be nested arbitrarily deep in a query. Hence the sequence of operators that are involved in calling a graph algorithm is:

1. CONSTRUCT-WHERE
2. INVOKE
3. PRODUCING
4. SELECT-WHERE

NOTE: As mentioned earlier, the graph that is created by the CONSTRUCT clause that is part of a CONSTRUCT-INVOKE-PRODUCING combination is never produced as output of the query; it is thrown away after it is used as input to the graph algorithm. If you want to see the graph that this CONSTRUCT clause builds, you must write a separate CONSTRUCT query.

Example: Sequence of Operators

The following example illustrates the use of both spam and non-spam vertices with Bad Rank:

```
PREFIX cray: <http://cray.com>
SELECT ?vertex ?ranking {
  CONSTRUCT {
   ?sub ?pred ?obj .
  } WHERE{
       {
         ?sub <http://bgf/isLinked> ?obj .
         ?sub <http://bgf/hasWeightLink> ?weightURI .
         ?obj <http://bgf/hasWeightLink> ?weightURI .
         ?weightURI <http://bgf/hasWeight> ?pred
       } UNION {
         ?sub <http://bgf/hasClassification> <http://bgf/spam> .
         BIND (<http://bgf/hasClassification> as ?pred) .
         BIND (<http://bgf/spam> as ?obj)
       } UNION {
         ?sub <http://bgf/hasClassification> <http://bgf/nonspam> .
         BIND (<http://bgf/hasClassification> as ?pred) .
         BIND (<http://bgf/nonspam> as ?obj)
       }
   }
  INVOKE cray:graphAlgorithm.badrank (0.0001, .84, 0.01,
  <http://bgf/spam>, <http://bgf/nonspam>, <http://bgf/hasClassification>)
  PRODUCING ?vertex ?ranking
}
ORDER BY DESC (?ranking)
LIMIT 100
```

Betweenness Centrality

URI and scalar arguments

```
<http://cray.com/graphAlgorithm.betweenness_centrality> (st_vx_ct, normalize)
```

In the above URI, `st_vx_ct` and `normalize` are used as examples and are explained later in *Inputs and Default Values*.

Description

This is the CGE-specific implementation of the classical vertex-betweenness-centrality algorithm. This algorithm assigns each vertex a numerical score. Take a given vertex V. In full generality, its betweenness score is defined to be the sum (over all other pairs of vertices) of the ratio of the number of shortest paths between that pair that go through V, over the total number of shortest paths between that pair. Thus it measures a sort of “importance” of each vertex, in terms of the shortest paths to other vertices that pass through it.

Inputs and Default Values

- **Vector inputs** - None
- **Scalar inputs** - None
- **Input Graph** - The input graph to the Betweenness centrality function is expected to contain triples of the form (vertex1, weight, vertex2) where weight is an integer. The following table describes the inputs that may be provided to the `INVOKE` keyword to invoke the Betweenness Centrality function.

Parameter	Description	Default Value
`st_vx_ct`	The *`st_vx_ct`* parameter can either be an integer or a decimal. ● If the *`starting_vertex_ctl`* parameter is an integer, it represents how many starting vertices should be used when approximating the betweenness score of every vertex in the graph. ● If the *`starting_vertex_ctl`* parameter is a decimal, it should be between `0.0` and `1.0`. If a decimal argument is used, the decimal value will represent the fraction of the graph's vertices, randomly chosen, that will be used as starting vertices for approximating the betweenness scores. A value of `1.0` (the default) specifies that every vertex in the graph will be used as a starting vertex.	`1.0`
`normalize`	The *`normalize`* parameter specifies whether or not the betweenness scores should be normalized. The acceptable values for this parameter are `0` and `1`, where `1` specifies that betweenness scores should be normalized. Normalizing the scores means to subtract from the betweenness score of each vertex the minimum betweenness score and then divide that partial result by the difference between the maximum and minimum betweenness scores found among all the vertices. Normalized scores will be between `0.0` and `1.0`.	`1`

Outputs

A call to the Betweenness Centrality function returns a two-column intermediate result set. The first column contains the vertex identifier (URI), whereas the second column contains the centrality score of the vertex. In other words, each row of the output result set pairs a vertex's ID with a double-precision floating-point value representing the centrality score for that vertex.

Example: Betweenness Centrality

```
PREFIX cray: <http://cray.com>
SELECT ?vertices ?scores
WHERE {
  CONSTRUCT {
     ?sub ?pred ?obj .
  } WHERE{
      ?sub ?pred ?obj .
  }
    INVOKE cray:graphAlgorithm.betweenness_centrality(.01,1)
    PRODUCING ?vertices ?scores
}
ORDER BY DESC(?scores)
```

S-T Set Connectivity

URI

```
http://cray.com/graphAlgorithm.st_set_connectivity
```

Inputs and Default Values

- Scalar inputs - None.
- Vector inputs - The S-T Set Connectivity function accepts input of a set of vertices designated as sources and a set of vertices designated as targets. These are added to the constructed graph using the `<http://cray.com/sourceVertex>` and `<http://cray.com/targetVertex>` URIs, as well as the standard RDFS predicate <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, which may be abbreviated as “a” in a SPARQL query.

Subject	Predicate	Object
Source vertex identifier	`a`	`<http://cray.com/sourceVertex`
Target vertex identifier	`a`	`<http://cray.com/targetVertex>`

Outputs

A call to the S-T Set Connectivity function returns an intermediate result set containing a single integer. The values and meaning of this integer are described below:

- If the integer’s value is `0`, there is no path between any pair of vertices with the source vertex taken from the source set and the target vertex taken from the target set.
- If the value is greater than `0`, it represents the number of hops in the shortest path between any such pair of vertices.

IMPORTANT: The S-T Set Connectivity function will return an error in the following cases:

1. Nonexistence of input source and/or target vertex
2. Invalid input source and/or target vertex
3. Nonexistence of input source and/or target vertex in the input edge list

Example: S-T Set Connectivity

The following example selects all of the edges from the default graph from the default graph and calls S-T Set Connectivity on the resulting graph.

```
PREFIX cray: <http://cray.com/>
SELECT ?distance
WHERE {
  CONSTRUCT{
     ?sub ?pred ?obj .
     ?srcNode a cray:sourceVertex .
     ?trgNode a cray:targetVertex .
    }
  WHERE{
     {
        ?sub ?pred ?obj .
     }
```

```
      UNION {
         VALUES ?srcNode
         {
           <http://bgf.org/c/03/i/000000>
           <http://bgf.org/c/05/i/000000>
           <http://bgf.org/c/08/i/000003>
         }
      }
      UNION {
         VALUES ?trgNode
         {
           <http://bgf.org/c/05/i/000001>
           <http://bgf.org/c/08/i/000007>
         }
      }
   }
   INVOKE cray:graphAlgorithm.st_set_connectivity()
   PRODUCING ?distance
 }
```

S-T (Source – Target) Connectivity

URI

```
<http://cray.com/graphAlgorithm.st_connectivity>
```

Description

The S-T Connectivity function calculates the length of the path between two vertices, if one exists.

Inputs and Default Values

- **Vector inputs** - None.
- **Scalar inputs** - The input graph to the S-T Connectivity function is expected to contain triples of the form (vertex1, predicate, vertex2) where the value of predicate is ignored. The S-T Connectivity function requires two scalar input arguments, which are the IRIs of the two vertices under consideration, source and target, respectively. This is illustrated in the example below:

  ```
  INVOKE <http://cray.com/graphAlgorithm.st_connectivity>
  (<urn:mySourceVertex>, <urn:myTargetVertex>)
  ```

  In the above example, `<urn:mySourceVertex>` and `<urn:myTargetVertex>` are the IRIs of the source and target vertices, respectively.

Outputs

The following example culls needed edges from the default graph and calls S-T Connectivity on the resulting graph.

Example: S-T Connectivity

```
PREFIX cray: <http://cray.com/>

SELECT ?nHops
WHERE {
```

```
   CONSTRUCT {
     ?v1 ?p ?v2 .
   } WHERE {
     SELECT ?v1 ?v2 ?p
     WHERE {
       ?v1 <urn:hasLink> ?v2 .
       BIND(<urn:path> AS ?p)
     }
   }
     INVOKE cray:graphAlgorithm.st_connectivity(<http://ga.org/string#000/
vertex#00000001>,
         <http://ga.org/string#000/vertex#00200000>)
     PRODUCING ?nHops
}
```

Label Propagation Argument Descriptions

URI

```
<http://cray.com/graphAlgorithm.label_propagation>
```

Description

The Label Propagation algorithm is used for detecting communities in networks and assigns vertices in the graph to communities. Each vertex is initially assigned to its own community. At every step, each vertex looks at the community affiliation of all its neighbors, and updates their state to the mode community affiliation. The mode community affiliation takes into account the edge weights.

The Label Propagation algorithm is relatively inexpensive, but convergence is not guaranteed.

Inputs and Default Values

The input graph to the Label Propagation function is expected to contain triples of the form (*vertex1*, *weight*, *vertex2*), where *weight* is an integer.

Input	Default Value
The number of steps that the algorithm executes. Currently an early exit is not included if convergence is detected. Therefore, the algorithm executes the number of steps specified in the input.	20

Outputs

A call to the Label Propagation function returns an array of vertex IDs paired with an array of community IDs These IDs can be used to identify which community each vertex was assigned to.

Example: Label Propagation

```
PREFIX cray: <http://cray.com/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?vertex ?comm
WHERE{
CONSTRUCT {
     ?sub ?weight ?obj .
} WHERE {
     ?sub <http://wga/isLinked> ?obj .
     ?sub <http://wga/hasWeightLink> ?weightURI .
     ?obj <http://wga/hasWeightLink> ?weightURI .
     ?weightURI <http://wga/hasWeight> ?weight
}
INVOKE  cray:graphAlgorithm.label_propagation(5)
PRODUCING ?vertex ?comm
}
ORDER BY ?comm
```

Bad Rank Argument Descriptions

URI

```
<http://cray.com/graphAlgorithm.badrank>
```

Inputs and Default Values

- Scalar inputs - None.
- Vector inputs - The Bad Rank algorithm accepts input designation of a set of vertices that are known to be “bad”, and optionally a set of vertices that are known to be "trusted", within the main input graph. This code can take three scalar arguments of type double.

Input	Default Value
The threshold of the maximum difference between per-vertex Bad Rank results from successive iterations of the algorithm below, which the algorithm will terminate.	`0.0001`
The probability that the next step in a (random) walk will be followed.	`0.84`
The probability that a random walk will take a next step to a bad vertex.	`0.01`
The URI that designates the object field of a triple that identifies a spam vertex	<http://cray.com/spamVertex>
The URI that designates the object field of a triple that identifies a non-spam, or trusted vertex.	<http://cray.com/nonspamVertex>
The URI that designates the predicate field of a triple that identifies either a spam or a non-spam vertex.	Defaults to the standard RDFS type predicate, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> The above can be abbreviated in a SPARQL query as “`a`”.
The indicator that specifies whether or not normalization should be applied to results. Acceptable values for this parameter are `0` and `1`.	`1`. If the default value is used, the scores are all mapped to floating point numbers between `0.0` and `1.0`, with the maximum value found mapping to `1.0`, the minimum score found mapping to `0.0`, and other scores mapping between those values proportionately.

Input	Default Value
	If the value is set to 0, results will not be normalized and will be presented as Bad Rank computed them.

Outputs

Bad Rank produces a two-column intermediate result that can be thought of as a set of pairs. The first item in each pair is the identifier of a vertex, whereas the second is the double-precision Bad Rank value of the vertex.

Cray Graph Engine (CGE) Security Mechanisms

The CGE query engine protects the port on which it communicates with clients using an encrypted authentication mechanism based on the Secure Shell (SSH) passwordless authentication mechanism. Before using the CGE user interface query clients to make requests on data sets, authentication must be configured. If it is required to set up the query engine to permit multiple users to execute requests, it will be required to configure public keys for each user. This can be configured on a per-data set or all data sets basis.

Grant Basic Access to Owned Query Engines

About this task

The Cray Graph Engine (CGE) query engine and CGE CLI commands use your SSH configuration to obtain public and private keys for use in authentication. Configuring basic query engine authentication is almost the same as configuring SSH passwordless authentication to the localhost IP host for your login account. The steps involved in granting basic access to your query engine are listed below:

Procedure

1. Ensure that you have a `.ssh` directory in your home directory and that the directory permissions are `700 (rwx------).`

   To find out whether you have a .ssh directory, and whether or not it is correctly protected, use the following command:

   ```
   $ ls -ld $HOME/.ssh
   drwx------  6 username  group  204 Nov 20 07:15 /users/username/.ssh
   ```

   If this looks correct you can move on to the next step. If the directory does not exist at all, you will need to create it, as shown below:

   ```
   $ mkdir $HOME/.ssh
   $ chmod 700 $HOME/.ssh
   $ ls -ld $HOME/.ssh
   drwx------  6 username  group  204 Nov 20 07:15 /users/username/.ssh
   ```

   If the directory does not have the correct permissions, you can simply change those. However, it is important to ensure that the directory is writable only by you. As long as this requirement is met, you do not need to change anything. The following command can be used if it is requried to set the permissions on the directory:

   ```
   $ chmod 700 $HOME/.ssh
   $ ls -ld $HOME/.ssh
   drwx------  6 username  group  204 Nov 20 07:15 /users/username/.ssh
   ```

2. Create a public / private authentication key pair using `ssh-keygen` if the key pair does not currently exist. Use the following command to find out whether or not you already have a public / private key pair configured.

NOTE: The following shows only key files (you will probably have other files as well unless this is a brand new `.ssh` directory):

```
$ ls -l $HOME/.ssh
total 80
-rw------- 1 username group   668 Apr  8  2014 id_dsa
-rw-r--r-- 1 username group   601 Apr  8  2014 id_dsa.pub
-rw------- 1 username group   883 Apr  8  2014 id_rsa
-rw-r--r-- 1 username group   221 Apr  8  2014 id_rsa.pub
```

In the above example you may have only an RSA key pair (`id_rsa` and `id_rsa.pub`) only a DSA key pair (`id_dsa` and `id_dsa.pub`) or both. A file with ".pub" in its name is a public key file. A file without ".pub" in its name is a private key file. All of your private key files should have `-rw-------` for their permissions as shown above. Your public key files may be readable (not writable) by anyone, but do not need to be, so the permissions shown above are okay, but not required. The minimum permission set that should be used is `–rw-------` , this enables reading and modifying the file. The maximum permission set should have `-rw-r--r--` , which permits other users to read but not modify the public key. If there is not even a single public/private key pair in the `.ssh` directory, it will be needed to generate an SSH key. This can be done using the `ssh-keygen` command:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/users/username/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /users/username/.ssh/id_rsa.
Your public key has been saved in /users/username/.ssh/id_rsa.pub.
The key fingerprint is:
eb:0d:10:cd:4f:4b:f1:2b:20:87:99:82:93:b5:8d:ee [MD5] username@host
The key's randomart image is:
+--[ RSA 2048]----+
|    .     .      |
|   + + *   o     |
|  + + B = o .    |
|   o . + = . .   |
|    . . S + .    |
|   .   . . .     |
|    E   o        |
|       . o       |
|        . .      |
+--[MD5]----------+
$ ls -l $HOME/.ssh
total 8
-rw------- 1 username group 1679 Jan  6 11:49 id_rsa
-rw-r--r-- 1 username group  391 Jan  6 11:49 id_rsa.pub
```

This produces a public / private key pair which can be used for passwordless authentication to localhost.

NOTE: At present, CGE does not support `ssh-agent` forwarding, so we do not recommend that you specify a pass-phrase when creating your key.

3. Place your public authentication key in your `.ssh/authorized_keys` file. This will allow you to interact with CGE query engines started by you on this machine (it does not allow other users to use your query engines). Set this up as follows:

```
$ cat $HOME/.ssh/id_*.pub >> $HOME/.ssh/authorized_keys
$ chmod 644 $HOME/.ssh/authorized_keys
$ ls -l $HOME/.ssh
total 80
-rw-r--r-- 1 username group  2601 Jun 18  2014 authorized_keys
-rw------- 1 username group   668 Apr  8  2014 id_dsa
-rw-r--r-- 1 username group   601 Apr  8  2014 id_dsa.pub
-rw------- 1 username group   883 Apr  8  2014 id_rsa
-rw-r--r-- 1 username group   221 Apr  8  2014 id_rsa.pub
```

4. Test using ssh to log into localhost without a password. The simplest way to test this is to try connecting to localhost through SSH. This will cause you to log in to the same host you are logged into currently again:

```
$ ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is 0a:34:d6:d9:71:b4:6c:e6:1d:49:95:ea:7d:09:54:89 [MD5].
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Last login: Tue Jan  6 11:56:10 2015 from localhost
-------------------------------------------------------------------------------
Message of the day...
-------------------------------------------------------------------------------
$ exit
```

As you can see, the first time you do this, you will be prompted to verify that the key for localhost is correct. You will also be prompted like this the first time you try to connect with a query engine with a new TCP/IP port number, so it is a good idea to do an interactive query or other kind of front-end command before trying to use a new query engine port from a script or other automated environment. Once you have verified the authenticity of the host / port pair, this pair will be added automatically to your list of known hosts and you should not need to do this again.

NOTE: To avoid the need for performing the interactive Host Key verification step, see *Eliminating the Interactive Host Key Verification*

To show that this works, try a second attempt to use SSH to log into localhost:

```
$  ssh localhost
Last login: Tue Jan  6 11:56:10 2015 from localhost
------------------------------------------------------------------------------
-
Message of the day...
------------------------------------------------------------------------------
-
$ exit
$
```

5. Once you have set this up, you need to authenticate the `localhost / <port number>` pairs for all of your query engine ports so that the clients can connect non-interactively. To do this, start CGE on each port you intend to use and run an interactive request through CGE, once for each port. The `cge-cli echo` command provides a simple way of doing so, as shown below:

```
$ cge-cli echo --db-port=73737
The authenticity of host 'localhost' can't be established.
RSA key fingerprint is d2:b4:ad:70:f1:44:d3:8a:f5:16:db:db:76:07:19:47.
Are you sure you want to continue connecting? [Yes/No]: yes
13835 [main] WARN com.cray.cge.cli.communications.client.ssh.LoggingBridge  - Permanently added 'localhost' (RSA) to the list of
known hosts.
14110 [main] INFO com.cray.cge.cli.commands.debug.EchoCommand  - Sending echo request...
14157 [main] INFO com.cray.cge.cli.lightweight.commands.debug.EchoCommand  - Echoed data received and validated successfully
```

NOTE: To avoid the need for performing the interactive Host Key verification step, see *Eliminating the Interactive Host Key Verification*

Eliminating the Interactive Host Key Verification

The SSH protocol uses the host key to authenticate the server to the client, which is of particular importance when the client will be sending confidential data (passwords, for example) to the server. Since the SSH protocol used by CGE does not permit the use of passwords, and the clients do not generally send other secrets to CGE, there is no real need for the client (and the invoking user) to verify that the host key is the one that the user trusts.

By default, the CGE CLI commands require explicit first time verification of host keys, as you have seen in the examples above. There is, however, a setting that you can set in your `cge.properties file(s)` that will cause the CGE CLI commands to consider any host key as trusted. This eliminates the need for a first-time interactive CLI command each time you start using a server on a new TCP/IP port number, and streamlines the process of connecting to a new instance CGE.

To add this setting, make sure that all appropriate `cge.properties` files contain the following line:

```
cge.cli.trust-keys=true
```

The same behavior can be achieved by adding the `--trust-keys` option to any of the CGE CLI commands.

> **IMPORTANT:** While implicitly trusting host keys for CGE is generally a safe practice, in the case where your data set contains actual confidential data, and you are using the CGE CLI clients to update the data set with new confidential data, you want to be certain that there is nothing other than CGE itself listening to the contents of your updates. In that case, the host key is an important part of ensuring that there is nothing between you and your CGE instance. This is not expected to be a common case among CGE users, but if your use of CGE falls into this category, it is recommended not to use the mechanisms described here.

Create a CGE Specific RSA/DSA Host Key

About this task

At some sites, site policy may dictate the use of a pass phrase with SSH keys used for logging into a system. If a pass phrase is used when creating your SSH key, the CGE authentication mechanism will be unable to use your SSH key(s) as its host key(s), so separate CGE specific host key(s) will need to be created. To do this, follow the instructions listed below:

Procedure

Instead of creating the key in your `.ssh` directory, create the key in your `.cge` directory using `ssh-keygen(1)` as before:

```
$ mkdir -p $HOME/.cge
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/users/username/.ssh/id_rsa): /users/username/.cge/id_rsa
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /users/username/.cge/id_rsa.

Your public key has been saved in /users/username/.cge/id_rsa.pub.
The key fingerprint is:
eb:0d:10:cd:4f:4b:f1:2b:20:87:99:82:93:b5:8d:ee [MD5] username@host
The key's randomart image is:
+--[ RSA 2048]----+
|    .     .      |
|   + + *   o     |
|  + + B = o .    |
|   o . + = . .   |
|    . . S + .    |
|   .   . . .     |
|    E   o        |
|       . o       |
|        . .      |
+--[MD5]----------+
$ ls -l $HOME/.cge
total 8
-rw------- 1 username group 1679 Jan  6 11:49 id_rsa
-rw-r--r-- 1 username group  391 Jan  6 11:49 id_rsa.pub
```

Once this has been done, CGE will use the keys in the `.cge` directory instead of the ones in the `.ssh` directory and you should have no further problems with pass phrases.

Grant Other Users Access to Owned Query Engine

The Cray Graph Engine (CGE) can protect the contents of your data sets from view/modification by unauthorized users via CGE instances that you run. Regardless of this protection, you must protect the raw data in your data sets using traditional Linux file protection, otherwise users who have access to your data can start their own query engine, using your data without your knowledge. To ensure that only users you authorize gain access to your data, it is best to set the permissions on each directory containing a data set to permit access (read, write and execute/search) only by its owner, and then to set the permissions on the files in the directory to permit access (read and write) only to their owners.

As the owner of a running instance of a CGE, you can control the list of users to whom you grant access. There are two modes of granting access to other users:

- Access to a single data set
- Access to any data set you provide

A key first step to any of this is protecting your data sets from being used under some other user's instance of CGE. If a user can run her own instance of CGE using your data, then you have no further control. So, if you want to control access to your data sets, make sure they are protected against access by users other than you. By setting the permissions on the data directory for the data set to `rwx------` you achieve this by preventing other users from looking in that directory for files. If you don't care about other users running their own instances of CGE using your data, you may set these permissions any way you like.

Assuming you have protected your data sets against other users, now you can grant individual users access to them. Regardless of whether you want to grant access to one or all data sets, you need the contents of each user's public key file from that user's .ssh directory. The user can follow the steps for setting up keys shown above if she does not have them yet. It is okay for the user to send you the public key(s) via e-mail, or any other method (including letting you copy them from the files yourself). What you will do with them is append them to an appropriate `authorized_keys` file (as described below).

> **IMPORTANT:** Remember that any user trying to connect with your server will need to authenticate your server as described in *Grant Basic Access to Owned Query Engines* or configure the CLI to trust Host Keys as described in *Eliminating the Interactive Host Key Verification*.

Ask your users to do the following after you have granted them access:

```
$ cge echo --db-port=73737
The authenticity of host localhost' can't be established.
RSA key fingerprint is d2:b4:ad:70:f1:44:d3:8a:f5:16:db:db:76:07:19:47.
Are you sure you want to continue connecting? [Yes/No]: yes
13835 [main] WARN com.cray.cge.communications.client.ssh.LoggingBridge  - Permanently added 'localhost' (RSA) to the
list of known hosts.
14110 [main] INFO com.cray.cge.sparql.cli.lightweight.commands.debug.EchoCommand  - Sending echo request...
14157 [main] INFO com.cray.cge.sparql.cli.lightweight.commands.debug.EchoCommand  - Echoed data received and validated
successfully
```

> **NOTE:** It is important to note that you should NEVER add another user's public key to your `$HOME/.ssh/authorized_keys` file. Doing so will allow the user to login as you.

In the following example, it is assumed that `/lus/scratch/username/lubm0` contains one of your data sets:

```
$ ls -ld /lus/scratch/username/lubm0
drwxr-xr-x 2 username group 4096 Oct 20 14:23 /lus/scratch/username/lubm0
$ chmod og-rwx /lus/scratch/username/lubm0
$ ls -ld /lus/scratch/username/lubm0
drwx------ 2 username group 4096 Oct 20 14:23 /lus/scratch/username/lubm0
$ ls -l /lus/scratch/username/lubm0/
total 4796
-rw-r--r-- 1 username group     221 Jan  6 13:13 authorized_keys
-rwxr-xr-x 1 username group 3321856 Oct  9 11:52 dbQuads
-rwxr-xr-x 1 username group 1568768 Oct  9 11:52 string_table_chars
-rw-r--r-- 1 username group    8192 Oct  9 11:52 string_table_chars.index
$ chmod og-rwx /lus/scratch/username/lubm0/*
$ ls -l /lus/scratch/username/lubm0/
total 4796
-rw------- 1 username group     221 Jan  6 13:13 authorized_keys
-rwx------ 1 username group 3321856 Oct  9 11:52 dbQuads
-rwx------ 1 username group 1568768 Oct  9 11:52 string_table_chars
-rw------- 1 username group    8192 Oct  9 11:52 string_table_chars.index
```

Now this data set can only be used by instances of the query engine that you start yourself. Other users wanting access will need to connect with a client and will be subject to client authentication.

Grant Other Users Access to One of the Owned Data Sets

To grant a user access to one of your data sets, all you need to do is put the user's public key in the `anauthorized_keys` file in the same directory where your data set resides, as shown in the following example:

```
$ ls -l /lus/scratch/username/lubm0/
total 4792
-rwxr-xr-x 1 username group 3321856 Oct  9 11:52 dbQuads
-rwxr-xr-x 1 username group 1568768 Oct  9 11:52 string_table_chars
-rw-r--r-- 1 username group    8192 Oct  9 11:52 string_table_chars.index
$ cat my_friend_id_rsa.pub >> /lus/scratch/username/lubm0/authorized_keys
$ ls -l /lus/scratch/username/lubm0/
total 4796
-rw-r--r-- 1 username group     221 Jan  6 13:13 authorized_keys
-rwxr-xr-x 1 username group 3321856 Oct  9 11:52 dbQuads
-rwxr-xr-x 1 username group 1568768 Oct  9 11:52 string_table_chars
-rw-r--r-- 1 username group    8192 Oct  9 11:52 string_table_chars.index
$ cat /lus/scratch/username/built_lubm0/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAxp7+CpYHL44jmuWeGXEMy+ijE/
X72f70YL8neITsR5gotXCIZh9V0G9ar8mNDlkoshN7Jp1qiRrQjYNy93hs9BBCz9kA5V9PhGC59qypEhNovYRo48lsUvTmHK0RWOVLfIZKNCkLVmbQubmEzM0FfUoY/ifNbTfrV4yGH2PNA4k= my_friend@myhost
```

Once you have done this, the user '`my_friend`' will have access to this data set only and not to all of your data sets. You can copy the `authorized_keys` file to any other data set you want to grant access to, and edit it as needed.

Grant Other Users Access to All of the Owned Data Sets

If you do not need to restrict access to specific data sets to a particular user, it is simpler to grant that user access to all of your data sets in one `authorized_keys` file. CGE uses a directory located at `$HOME/.cge` that allows you to set up configuration files that apply to all of your data sets. By creating an `authorized_keys` file in this directory and putting authorized public keys in that file, you can grant access to all of your data sets, as shown in the following example:

```
% mkdir -p $HOME/.cge
$ chmod o-w,g-w $HOME/.cge
$ cat my_friend_id_rsa.pub >> $HOME/.cge/authorized_keys
$ ls -l $HOME/.cge
total 4796
-rw-r--r-- 1 username group     221 Jan  6 13:13 authorized_keys
$ cat $HOME/.cge/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAxp7+CpYHL44jmuWeGXEMy+ijE/
X72f70YL8neITsR5gotXCIZh9V0G9ar8mNDlkoshN7Jp1qiRrQjYNy93hs9BBCz9kA5V9PhGC59qypEhNovYRo48lsUvTmHK0RWOVLfIZKNCkLVmbQubmEzM0FfUoY/ifNbTfrV4yGH2PNA4k= my_friend@myhost
```

Now the user `my_friend` will have access to all of your data sets.

Cray Graph Engine (CGE) Extension Functions

CGE provides a number of extension functions, including:

- Interval analytics functions. See *Cray Graph Engine (CGE) Interval Analytics Functions* on page 70 for more information.
- Haversine functions. See *Cray Graph Engine (CGE) Haversine Functions* on page 74 for more information.
- Square root function. See *Cray Graph Engine (CGE) Square Root Function* on page 74 for more information.

Cray Graph Engine (CGE) Interval Analytics Functions

Intervals

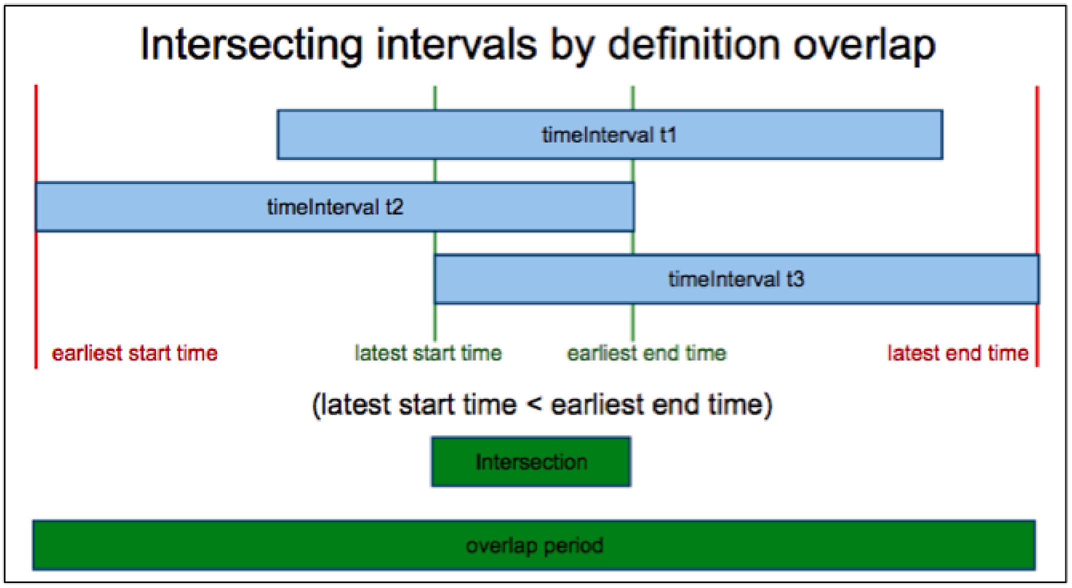
An interval is defined as the sequence between any two variables of compatible atomic types, where one defines the start of the interval and the other defines the end of the interval. The interval is inclusive of the start and end.

Intersecting and Non-Intersecting Intervals

Two or more intervals are said to be intersecting if there is an interval in time during which all of the intervals under consideration are present. More precisely, intersecting time intervals are those where the latest start time is less than the earliest end time, i.e., no period has ended before the last period to begin, has started. This period is termed as an intersection and starts at the beginning of the last interval to start and ends at the end of the first interval to end.

On the other hand, non-intersecting time intervals are those where a period has ended before the last period to begin has started, as illustrated in the following figure:

Figure 11. Intersecting Intervals



Continuous and Non-Continuous Intervals

Two or more intervals are said to be “continuous” if there is at least one interval present during the complete span, from the start of the first interval to start, to end of the last interval to end. In non-continuous intervals, at least one gap (period within which no intervals are present) is present between the intervals under consideration. This is illustrated in the following figure:

Figure 12. Continuous Period With Non-Intersecting Intervals

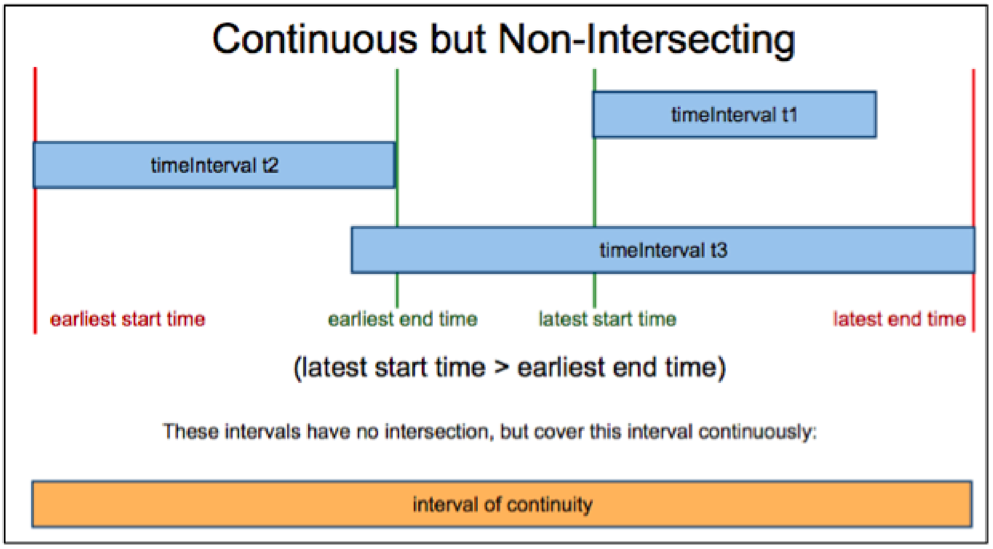
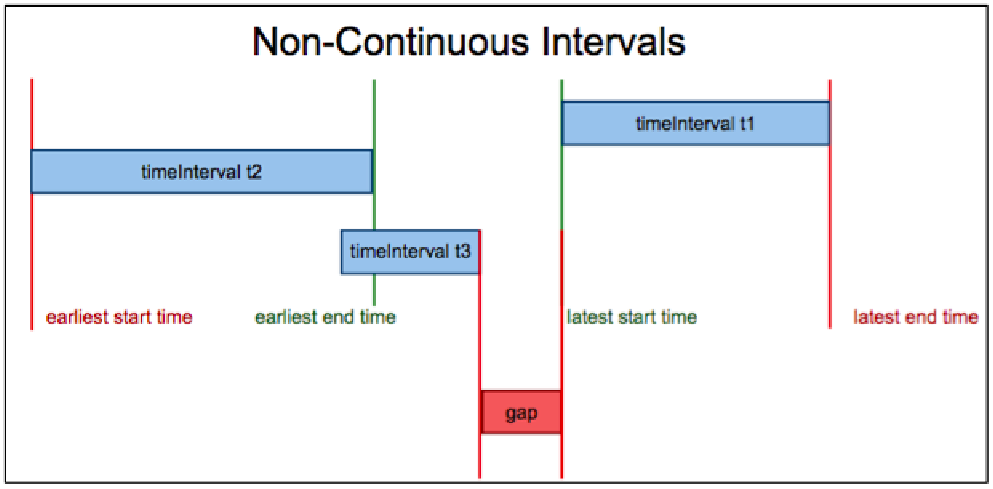

Continuous but Non-Intersecting
timeInterval t1
timeInterval t2
timeInterval t3
earliest start time
earliest end time
latest start time
latest end time
(latest start time > earliest end time)
These intervals have no intersection, but cover this interval continuously:
interval of continuity

Figure 13. Non-Continuous Period with Non-Intersecting Intervals



Using Interval Analytics Functions For Temporal Analysis

Interval functions can be used to gather fine-grained detail about intervals. For example, they can be used to:

- Determine if a time period that ends at the same time is contiguous with one that starts at the same time.
- Determine whether or not two or more time intervals intersect.
- Determine the continuity of a given time period.

Function Prefix

The prefix to use when using interval functions in queries is:

```
PREFIX arq: <http://jena.hpl.hp.com/ARQ/function#
```

List of Interval Analytics Functions

NOTE: The names of all the CGE interval functions are case-sensitive. Interval functions work with any type that has a < comparison, e.g., numerics and strings.

The list of CGE-specific interval analytics functions, their syntax and description is provided in the following table:

Table 12. List of CGE-specific Interval Functions

Function	Description
`listmin(element1, .... elementN)`	This function returns the smallest item in the comma-separated list of items provided as arguments.
`listmax(element1, .... elementN)`	This function returns the largest item in the list of arguments.
`iscontinuous(start1,end1,... startN, endN)`	This is a pairwise function that accepts a list of comma-separated list of start and end times and determines whether or not there is a gap between the intervals under consideration.

Function	**Description**
	● `True` when there is complete coverage from earliest starting time to latest end time, i.e. there are no gaps in the coverage. ● `False` if there is any gap in the coverage
`isintersecting(start1, end1, .... startN, endN)`	This is a pairwise function that determines whether or not there is a period within which all the intervals under consideration are present. This function returns: ● `True` when there is an interval where all intervals are present. ● `False` if there is no interval when all intervals are present
`duration(startTime, endTime)`	This function uses the Unix epoch and time functions to calculate the duration between the start and end times, which are provided as arguments. This function returns the xsd:dayTimeDuration between startTime and endTime. **NOTE:** This function only accepts dates starting from July 5, 1776.

NOTE: Although the `listmin()`, `listmax()`, `iscontinuous()` and `isintersecting()` functions support all SPARQL compatible types, the arguments provided to these function should all be of compatible atomic types, otherwise an `xsd_error` will be returned. Furthermore, the `duration()` function will return an `xsd_error` in the following cases:

- Either of the arguments are not of type `xsd:dateTime`
- The sum of (duration(`xsdDate1`, `xsdDateTime2`) - duration(`xsdDateTime2`, `xsdDate1`)) will not be zero. This is because `xsdDate` is defined to span 24 hours (for standard days), and it is assumed that the start time is at the beginning of the day, and the end time is at the end of the day

There are a few important items to note when using the interval analytics functions:

- The interval analytic functions do not fully support the `xsd:date` and `xsd:time` data types and may return incorrect results; users should avoid these two types.
- Comparisons of `xsd:date` and `xsd:dateTime` within the same day may return unexpected results. `xsd:date` and `xsd:dateTime` comparisons are supported outside of the 14 hour time zone range and the 24 hour day span of `xsd:date`.
- `xsd:date` results are now included when filtering on `xsd:dateTime` (outside the same day) and vice versa (`xsd:dateTime` results when filter on `xsd:date`). If strict `xsd:dateTime` results (or `xsd:date` results) are required, the appropriate data type filter should be added.

- The `duration()` function supports combinations of xsd:date and `xsd:dateTime.` If an `xsd:date` result is the start time, the duration will start at the beginning of the day. Similarly, if the `xsd:date` result is the end time, the duration will end at the end of the day.

Cray Graph Engine (CGE) Haversine Functions

CGE supports the haversinemeters() and haversinemiles() functions to enable support for spatially aware applications. These functions are based on the Haversine formula, which is an equation that calculates the great-circle distance between two points on a sphere from the longitudes and latitudes of the two points. For more information, visit *http://en.wikipedia.org/wiki/Haversine_formula.*

The syntax of CGE Haversine functions is shown below:

- `afq:haversinemeters(latStart, longStart, latEnd, longEnd)`
- `afq:haversinemiles(latStart, longStart, latEnd, longEnd)`

NOTE: The `haversinemeters()` and `haversinemiles()` functions are case sensitive.

Inputs

Both the CGE `haversinemeters()` and `haversinemiles()` functions accept the following inputs in `xsd:decimal`, `xsd:double` and `xsd:float` formats:

- **`atStart`** – The starting position of the latitude (dimensions of the values in degrees)
- **`longStart`** – The starting position of the longitude (dimensions of the values in degrees)
- **`latEnd`** – The ending position of the latitude (dimensions of the values in degrees)
- **`longEnd`** – The ending position of the latitude (dimensions of the values in degrees)

Acceptable latitude values range from `-90` to `90`, whereas acceptable longitude values range from `-180` degrees to `180` degrees.

NOTE: Important: The functions will return an empty value if:

- Invalid position coordinates are provided
- Empty input values are provided
- Insufficient parameters are provided.

Output

The `haversinemeters()` function returns the distance between two points in meters, whereas the `haversinemiles()` function returns the distance between two points in miles.

Function Prefix

The prefix to use when using CGE Haversine functions in queries is:

```
PREFIX afq: <http://jena.hpl.hp.com/ARQ/function#>
```

Cray Graph Engine (CGE) Square Root Function

The square root function, `sqrt()` is used to retrieve the square root of the specified number

Syntax

The syntax of the square root function is:

`sqrt(argument)`

> **NOTE:** The name of the `sqrt() function` is case sensitive.

Function Prefix

The prefix to use when using the `sqrt() function` in queries is:

`PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>`

Table 13. CGE Square Root Function's Examples

Argument Type	Example

`Integer`	`PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>`
	`SELECT ?a { BIND (afn:sqrt("9223372036854775807"^^ <http://www.w3.org/2001/XMLSchema#integer) AS ?a) }`

`Decimal`	`PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>`
	`SELECT ?a  { BIND (afn:sqrt(4294967296.0) AS ?a) }`

`Float`	`PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>`
	`SELECT ?a  { BIND (afn:sqrt ("3.4E38"^^xsd:float) AS ?a) }`

`Double`	`PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>`
	`SELECT ?a  { BIND (afn:sqrt("1.797E308"^^xsd:double) AS ?a) }`

`Boolean`	`PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>`
	`SELECT ?a  { BIND (afn:sqrt(true) AS ?a) }`
	NOTE: Passing “`true`” as the Boolean argument returns `1`, whereas passing “`false`” as the Boolean argument returns `0`.

> **NOTE:** The `sqrt()` function will return an empty value if a negative number is provided as an argument. Furthermore, the `sqrt()` function will return an empty value if arguments of certain types are used. These argument types include:
>
> - `xsd:dateTime`
> - String
> - IRI
> - Arbitrary data type

You can also use derived data types as arguments to the `sqrt() function`, as shown in the following query:

```
PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>
SELECT ?a{ BIND (afn:sqrt  ("18446744073709551615"^^<http://www.w3.org/2001/XMLSchema#positiveInteger>) AS ?a) }
```

> **NOTE:** Executing the `sqrt()` function when a negative derived type is used as an argument will result in an empty value.

Cray Graph Engine (CGE) Property Path Support

CGE does not natively support the SPARQL 1.1 property paths feature, however it does support certain types of property paths.

> **NOTE:** CGE's property path support should be used with care. This support is disabled by default and must be explicitly enabled by the user. Contact Cray Support for additional information.

- **Simple Property Paths** - By default simple property paths that are equivalent to simple fixed length Basic Graph Patterns (BGPs) are supported, this means that property paths consisting of only the sequence `/` and inverse `^` operators are permitted since these can be written out as a simple BGP using blank node variables. For example:

  ```
  SELECT * WHERE
  {
  ?s <urn:a>/<urn:b> ?o
  }
  ```

  Can be rewritten as follows:

  ```
  SELECT * WHERE
  {
  ?s <urn:a> _:p0 .
  _:p0 <urn:b> ?o .
  }
  ```

- **Complex Property Paths Emulation** - Some more complex property paths can be emulated through query rewriting, which expands the property paths into an equivalent query form.

  > **NOTE:** It is important to be aware that this support is only emulation, and may not provide complete answers that a SPARQL engine with native property path support would produce.

The following table details the additional operators, which may be emulated and the restrictions and limitations on that emulation.

Table 14. Additional Operators that May be Emulated

Operator	Example	Description	Additional Notes
`*`	`?s <urn:a>* ?o`	Finds paths of zero or more steps between two nodes in the graph	● Path to which the * operator applied **must** be either a predicate or inverse predicate ● Evaluating the zero length portion of the path may be very expensive particularly if both variables are unbound ● Paths are evaluated only up to a maximum length (default 5) which may be user configured on a per-query basis

Operator	Example	Description	Additional Notes
			• Expands into a UNION that looks for paths of each length up to the specified maximum
+	`?s <urn:a>+ ?o`	Finds paths of one or more steps between two nodes in the graph	• Path to which the + operator applied **must** be either a predicate or inverse predicate • Paths are evaluated only up to a maximum length (default being 5) which may be user configured on a per-query basis • Expands into a `UNION` that looks for paths of each length up to the specified maximum
?	`?s <urn:a>? ?o`	Finds paths of zero or one steps between two nodes in the graph	• Path to which the ? operator applied **must** be either a predicate or inverse predicate • Evaluating the zero length portion of the path may be very expensive particularly if both variables are unbound • Expands into a `UNION` that looks for paths of length zero and one
|	`?s <urn:a> | <urn:b> ?o`	Finds paths between two nodes that use any of the alternative paths given	• Paths to which the | operator applied may themselves be complex but only paths that are predicates or inverse predicates are guaranteed to expand into a valid query • Expands into a `UNION` that considers each alternative, where the alternative is itself a property path it may be further expanded as necessary
! (*property*)	`?s ! <urn:a> ?o`	Find paths between two nodes that **do not** pass through a given predicate	• The negated property set operator only applies to predicates or inverse predicates and thus can always be expanded • Expands into a `MINUS` that considers all paths and then eliminates the undesirable paths

Enabling Emulation

CGE also provides the option to change the maximum length of paths (for the expansion of the * and + operators), as shown in the following example:

```
% cge-cli query --opt-on optPathExpand --path-expansion 3 paths.rq
```

The above query would run the query with property path expansion enabled and a maximum path length of 3.

NOTE: This value can be set to any desired value, however it is important to note that the higher this value is set to, the more complex the query that will be generated. This will result in slower performance because the database server will need to search for longer paths. Therefore, it is recommended to set the length of paths to the minimum possible value for optimal emulation performance. It is also important to note that setting a maximum length of zero or less will result in disabling the expansion.

Logging and Troubleshooting

CGE produces a text log, which is a trace of program execution during query or update processing. Users can view the log with a text editor (such as vi), or typically the Linux `less` command. The log can be searched using the `grep` command for text messages of interest.

`INFO` messages will be deposited into the log during normal operation. CGE can also generate `ERROR` and `WARN` messages. All of these messages can yield information about activity that takes place during command execution.

System error message can be present in the log under conditions where CGE exits or improperly shuts down.

When queries or updates are executed, `INFO` messages with “`now starting query #`” are written to the log. For example:

```
2015-Feb-10 19:34:26.513 CST INFO [][7720] 0x43 parser/parseAndBuildSM.cpp@374 allocQueryGlobals [] [QRY ]  <OT> now starting query # 1
```

Many other `INFO` messages will also be deposited to the log during normal operation. For example, long processing times can be seen in the log from one `INFO` message to the next:

```
2015-Feb-13 14:44:45.500 CST INFO [][9448] 0xb utils/malloc/cqe_malloc.cpp@901 LogRequest [] [QRY |MEM ] image 0 : request by "file: parser/qengine/database.cpp,
func: readFromDisk line: 989" of 69.849 MiB (0x45d9688) was filled. (0x10005200c80)
2015-Feb-13 14:49:31.099 CST INFO [][9448] 0xc parser/qengine/database.cpp@1141 readFromDisk [] [QRY |STRT] time to read in db of size 139.698 GiB (0x22ecb28000):
285.679279
```

When large datasets are used, the `INFO` message for the total start up time can be long, as shown in the following example:

```
2014-Dec-18 14:40:37.428 CST INFO [][25977] 0x5b parser/dbServer.cpp@1259 main [] [QRY |STRT|PERF] Total startup time: 1434.489315 seconds
```

The following are examples of `ERROR` messages that CGE can produce when query or update processing has failed:

1. `No such file or directory`
2. `No space left on device`
3. `Exiting because malloc of`
4. `Lookup failure for HURI`
5. `Invalid graph algorithm name`
6. `Exiting with status`
7. `Bad entry`
8. `Short read`
9. `Assertion`
10. `Realloc of`
11. `Error detected in Dispatcher`

It is recommend to search the log for the text: "`ERROR`" and contact Cray Support if problems are encountered in query or update processing.

The following are samples of `WARN` messages that can be produced. `WARN` messages are subjective in preceding errors in processing:

1. `huri was not found`

2. `directory not specified`
3. `not found in IRA`
4. `No valid quads in database`
5. `Invalid object for quad`
6. `Number of warnings found`
7. `Unsupported datatype`
8. `not in the dictionary`
9. `IRA huris not allocated`

Search the log for `WARN` messages and contact Cray Support if problems in query or update processing are suspected.

The following are examples of system error messages that CGE can produce when query or update processing has failed. Search the log for the last `INFO` messages and contact Cray Support if any of these follow:

1. `DUE TO TIME LIMIT`
2. `terminate called without an active exception`
3. `srun: error`
4. `Segmentation fault`
5. `Bus error`
6. `free invalid pointer`
7. `Out of memory`
8. `Unable to terminate gracefully`
9. `Floating point exception`
10. `Aborted`
11. `Killed`
12. `Unable to allocate resources`
13. `Exited with exit code`
14. `Requested nodes are busy`
15. `transaction completed with an error state`
16. `LIBDMAPP ERROR`
17. `IRI Resolution Error`
18. `rpn not found for`
19. `Trapped with SIGINT`

Troubleshooting Common Cray Graph Engine (CGE) Issues

The most common errors that are likely to be encountered while using CGE involve failure to connect to a database server successfully. There are a variety of different errors that can occur depending on exactly what goes wrong with the communications between the CLI and the database server. Common error messages that are likely to be encountered along with troubleshooting techniques are documented in the following table.

NOTE: The mono-spaced text in the error messages column represents environment specific values that will be displayed in the error messages.

Table 15. CGE Error Messages and Troubleshooting Information

Error Message	Description	Resolution
`Unable to establish a connection to the database server at host:port as it does not appear to be running`	The CLI tried to connect to a database server running on the given host and port combination but was unable to establish a connection. This typically means one of two things: **1.** There is no database server running on that host and port **2.** Firewall rules are preventing access to that host and port	● Verify that you have passed the correct host and port to the CLI ● Verify that there is a database server running on that host and port ● Verify that there are no firewall rules that are preventing access to the host and port. Contact a system administrator for additional information.
`Unable to authenticate to the database server at host:port. You do not have any SSH keys present in your configured identity idDirectory`	The CLI tried to connect to a database server running on the given host and port combination. A connection was established successfully, but authentication to the database server failed because there are no SSH keys configured.	Create at least one SSH key and place it in the appropriate directory. For more information, see *Cray Graph Engine (CGE) Security Mechanisms* on page 64.
`Unable to authenticate to the database server at host:port. Your SSH key(s) from your configured identity directory are not in the authorized_keys file of the database or its owner`	The CLI tried to connect to a database server running on the given host and port combination. A connection was established successfully but authentication to the database server failed because none of the SSH keys were in the `authorized_keys` file that the database is using.	● Review the database logs (if possible) to see which `authorized_keys` file was in-use: ○ If the database server was launched, then this is either in the database directory itself or in the `~/.cge` directory

Error Message	Description	Resolution
		○ If another user launched the database server, contact them to find out which `authorized_keys` file is in-use ● Add the public key to the relevant `authorized_keys` file, or ask the relevant user to do so.
`Host key for host host:port is not trusted, please run in interactive mode and trust this key or manually add the host key to your known_hosts file in your configured identity idDirectory`	The CLI tried to connect to a database server running on the given host and port combination. A connection was successfully established but the database server was unable to prove its identity to the CLI because the host key provided by the database server was not trusted. This error is usually only seen the first time when a connection to a specific server instance is established. Once the key is trusted (see resolution steps) this error should no longer be seen for this host and port combination.	● If CGE is being run in interactive mode, the system will prompt to trust the host key. Enter `Yes` to do so. ● If it is required to use CGE non-interactively, adding the `--trust-keys` option to commands will automatically trust previously unknown host keys
`Timed out attempting to establish a database connection (waited N seconds), database server may be too busy to service your request currently`	The CLI tried to connect to a database server running on the given host and port combination but was unable to establish a connection within the timeout interval. This means that the database server is currently busy processing another request and cannot accept the request at this time.	● Check the database logs to see what the database is currently doing ○ If the last log message states: "`Trying to read RPN message from network...`" then the database is ready, otherwise the database is busy ● If the database is busy, there are a number of options that can be used to troubleshoot the issue: ○ Execute the request again later ○ Increase the timeout with the `--timeout` option to wait for longer

Error Message	Description	Resolution
		○ Disable the timeout by setting `--timeout 0` to wait indefinitely until the database server is ready to process the next request ● In rare cases, the database may have become hung (if it is busy and you have not see any new log messages for long periods of time then this is most likely the problem) in which case you should kill and restart the database server and then retry your commands

If you are consistently receiving the same error even after following the suggested fixes in this section we recommend that you add the --trace option to your command in order to get detailed information about the communications being attempted and review the log messages carefully both on the front end and server side to try and understand what is going wrong. It is also worth reviewing the server logs if you are able as there are some situations which will manifest as client side communications errors that can be down to the configuration of the server. For example if running the CLI in secure mode you may get client side errors that state a secure connection could not be established, reviewing the server logs might show you that your public key is not authorised for the database server or that the server is not running in secure more. If you are still unable to resolve the issue please contact Cray support providing logs from both the CLI and the Database Server to aid in diagnosis of the issue.