Cray® Graph Engine User Guide

(3.2.UP02)

S-3014

Contents

1 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.

Release Information

This publication version addresses the product version `3.2UP02` of the Cray® Graph Engine.

Record of Revision

Date	**Addressed Release**
September 2018	3.2UP02
May 2018	3.2UP01
December 2017	3.1UP02
November 2017	3.1UP01
April 2017	3.0UP00
December 2016	2.5UP00
August 2016	2.0UP00
March 2016	1.0UP00
March 2015	Beta release

Record of Revision

Includes updates to UI related sections and information about the `output` parameter that has been added with this release.

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.

Trademarks

2 About the Cray Graph Engine (CGE)

CGE is a highly optimized software application designed for 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.

2.1 CGE Features

Major features of CGE are listed below:

- An optimized query engine for high-speed parallel data analysis.
- Support for submitting queries, updates and creating checkpoints.
- A rich 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.
- Support for SPARQL aggregate functions.
- Multi-user support.
- Capability to cancel queries.
- Compatibility with POSIX-compliant file systems.
- Database preprocessing to apply inference rules to the data, as well as to index the data.
- CGE Python, CGE Java and CGE Spark APIs
- Support for a number of built in graph algorithms.

2.2 Concepts of Operation

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

- The graph oriented database

- Resource Description Framework (RDF)

2.2.1 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
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

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 schema.

2.2.2 Differences Between CGE and 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
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

As opposed to relational databases, CGE uses RDF to store data.

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 using Uniform Resource Identifiers (URIs) in order to create triples to name a subject, an object, and the relationship or predicate between the two.

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)
```

The three statements within the preceding code block should be entered on a single line and have been shown in separate lines in this publication 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.

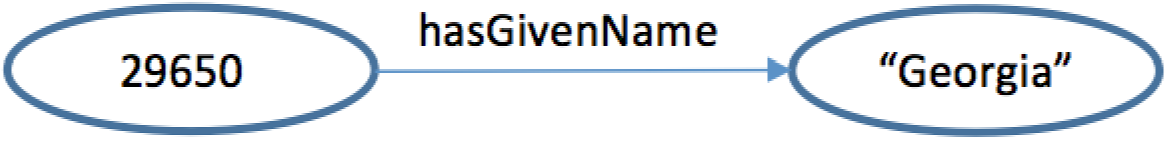
RDF triples are intended to be self-identifying in two ways, both of which can be seen in this example:

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.

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>
```

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 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. CGE supports the capability of nesting a call to a classical graph analysis function within a SPARQL query for indefinite pattern sizes and aggregate information that can not be expressed in SPARQL.

Each subject-predicate-object 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.

2.3 About SPARQL

SPARQL is an RDF query language developed for executing 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.

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.

  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.

2.4 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.

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.

2.5 RDF and SPARQL Resources

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

RDF Resources

- RDF primer at *https://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 *https://www.w3.org/TR/sparql11-query/*

Semantic Web Resources

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

3 The CGE Database Build Process

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 with the same database directory will use the compiled database. The `update` command can then be used to add data to an existing database or to update it. For more information, see the `cge-launch` and `update` man pages.

Data must be presented in this directory in one of the following ways to enable CGE to recognize raw RDF data to be built:

1. In a single file called `dataset.nq` (for N-Quads form data)
2. In a single file called `dataset.nt` (for N-Triples form data)
3. In multiple files listed in a file called `graph.info`

Data to RDF Triples Conversion

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.

Internal Representation

Once the data has been translated into RDF, the user must place the data in the directory where CGE builds its compiled database files. If the RDF is contained in a single file, 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 that has been extracted from the Lehigh University Benchmark (LUBM) synthetic dataset:

```
<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> .
```

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

The specification for NTriples can be found at *https://www.w3.org/TR/n-triples/*

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>
```

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.

⚠ **WARNING:** The `-d` parameter is mandatory. Launching CGE without specifying it will result in 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.

CGE can begin executing queries and updates once the database has been built. 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.

⚠ **CAUTION:** If a user attempts to create a new database and the input data files do not contain any valid triples, the database will exit with an error. The recommended way of creating an empty database is to create a completely empty input file using the `touch` command and then starting the database.

CGE searches for a dataset in the following places when loading a dataset:

- If `dbQuads` exists, it will be used.

- If `dbQuads` does not exist, but `graph.info` exists, `graph.info` will be opened and read to obtain a list of source data files, which will then be used to build a new dataset.
- If neither `dbQuads` nor `graph.info` exist, but `dataset.nt` (or `dataset.nq`) exist, `dataset.nt` or `dataset.nq` will be used to build a new dataset.
- If none of the above files exist, CGE will fail.

In each of these cases, if the file exists but is in some way invalid, CGE will fail.

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 4 times the size of the RDF file(s). For example, for a 100 GiB triples file, at least 400 GiB (4 * 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.

3.1 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***).

CGE's rules inference engine does not implement backward chaining, but it implements a highly parallel form of forward chaining.

3.2 About Inference Rules Files

Inferencing can be performed to generate additional relationships once the 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 Apache Jena rules.

In this version of CGE there are certain limitations to these rules:

- 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.

⚠ **CAUTION:** 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.

Inference Rules File Format

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 the list of limitations above. 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.

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.

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

Examples

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

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 and 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.

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 and 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 American based database.

3.3 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.

# [transitivePropery1: (?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.

3.4 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.

4 Launch the CGE Server Using the cge-launch Command

The cge-launch command launches the query engine and enables creating and building a database in a single step. It handles the details of allocating batch resources, setting up the launch environment, and composing a command line for the query engine on a given platform.

Following is an example of using cge-launch:

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

In the preceding statement:

- *pathtoResultsFile* - specifies the directory that will contain the results of queries and/or updates
- *path* - specifies the path to the database directory
- *logfile* - specifies the log file that will contain the command and server output.

The following options of the cage-launch command must be specified when launching the server:

- The -d option that specifies the path to the directory where the data set resides.
- The -o option that specifies the path to a directory where the result files produced by queries need to be placed.

Both the -d and -o options accept:

- UNIX style pathnames as naming files on a POSIX compliant file system
- URLs of the following forms:
  - file://*unix_pathname* - This form is the equivalent of the Unix Style Pathname in URL form
  - hdfs://*name-server-address*[:*name-server-port-number*]/*HDFS_pathname* - This type of format indicates that a Hadoop Distributed File System (HDFS) file or directory is known to the specified name server and is located within that name-server's name space at *HDFS_pathname*.

  Both the aforementioned forms must refer to a file/directory that is shared across and equally accessible from all nodes. CGE will determine where to look for this file/directory based on recognizing one of the aforementioned path formats.

When using checkpoints:

- If a full URL is used, the checkpoint is written exactly as specified by the URL, which means that an HDFS URL will cause the checkpoint to be written to the path specified in the URL on the HDFS file system described by the rest of the URL, and a file URL (i.e. file:/path) will be written to the POSIX file system at the pathname specified in the URL.
- If a relative path (i.e. a simple path with no leading / character) is used, the checkpoint will be written in a directory relative to the data directory used at CGE start up.
- If a full pathname but not a URL is specified, the pathname will be interpreted within the space specified by the URL of the data directory used at CGE start up, so, if CGE was started using an HDFS URL, the

checkpoint will be written at the specified path within HDFS, if CGE was started with a simple pathname or file URL, the checkpoint will be written at the specified path within the POSIX file space.

TIP: Relaunch CGE if the system displays an error message saying, "`Server failed to start up`" upon execution of the `cge-launch` command.

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

5 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)

5.1 CGE CLI

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 default JVM's maximum memory allocation when launching `cge-cli` is 2GB.

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
```

Table 1. CGE CLI Commands and Descriptions

Command	Description
`cge-cli checkpoint`	Requests checkpoint creation.
`cge-cli echo`	Allows sending echo requests, which can be used to ping CGE.
`cge-cli fe`	Launches a web-based interface for accessing the server and provides SPARQL endpoints, which can be accessed via standard SPARQL APIs and tooling.
`cge-cli help`	Displays help information.
`cge-cli get-configuration`	Determines the locations being searched for configuration files and effective properties.
`cge-cli keyword-lookup`	Provides help with converting keywords between names and indexes to help determine the log options to use with other commands.
`cge-cli log-info`	Retrieves the server's current logging setup.
`cge-cli log-lookup`	Provides help with converting log levels between names and values to help determine the log options to use with other commands.
`cge-cli log-reconfigure`	Reconfigures the default logging setup of the server. The logging configuration changes persist until the server is shut down.

Command	Description
`cge-cli nvp-info`	Retrieves the current NVP setup of the server
`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
`cge-cli generate keystore`	Creates/inspects a Java keystore file, which is used to enable SSL support for the `fe` command.
`cge-cli generate`	Generates a Shiro configuration template that can be customized as desired.
`cge-cli generate properties`	Creates a properties file that can be used to provide a variety of configuration to commands, without needing to specify it directly at the command line.

Use the `cge-cli help` command to retrieve help information for a specific CGE command, as shown in the following example:

```
$ cge-cli help command
```

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, as shown in the following example:

```
$ 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.

5.1.1 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.

5.1.2 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, several default locations are tried and the first valid location is used
`--keep-alive-timeout`	Seconds	60	`--keep-alive-timeout 30`	Configures the connection keep alive time out. As

Option	Argument(s)	Default Value	Example	Purpose
				of CGE 3.1UP00, connections are cached and kept alive. This results in improved performance, especially in situations that require many requests to be issued to the database.
`--no-keep-alive`	N/A	N/A	`--no-keep-alive`	Disables connection keep alive. This may be useful in multi user environments, where many users are sharing the same database
`--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.
`--username`	Username	`alice`	`--username alice`	When set, use this username to connect to the database. In order for this to work, it is required to have

Option	Argument(s)	Default Value	Example	Purpose
				access to a key pair which has been authorized for the given username. Therefore, this does not permit the user to impersonate arbitrary users, instead it allows the user to act as another user only if the user has an appropriate key pair.
Client Logging Options				
`--debug` `--verbose`	N/A	N/A	`--verbose`	Enables verbose mode, which includes setting the log level to `debug`. 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 sets the log level to `error`, causes little/no logging to go stderr. 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 trace mode, which includes setting the log level to `trace`. 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
`--reveal`			`--reveal`	Reveals user data in client side logging output. By default any logging that contains items considered to be user data e.g. Queries, query plans etc is obscured to prevent data leakage. Enabling this option disables that functionality.
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

Option	Argument(s)	Default Value	Example	Purpose
				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 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 log line pertaining to the request. This is useful if it is required to isolate and extract the log lines specific to a request.
`--log-keyword-level`	*`Keyword_level`*	N/A	`--log-keyword-level 41 32`	Changes the server 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

Option	**Argument(s)**	**Default Value**	**Example**	**Purpose**
				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.
`--configDir` `--config-dir`	Directory	N/A	`--configDir/`*`configpath`*	Sets the first location to search for configuration files
`--`	N/A	N/A	`--`	Used to separate the options from the arguments to the command.

Option	Argument(s)	Default Value	Example	Purpose
				This is useful if arguments may be mistaken for options. Any arguments seen after the `--` are treated as arguments even if they could otherwise be considered as options.

5.1.3 SSH Identities

SSH is used to encrypt communications with the database and to verify that a user has authorized access to the database. Use the `-i` or `--identity` options to specify an identity directory or directories. The following locations will be searched for keys in the absence of these options:

1. The `$CGE_CONFIG_DIR_NAME` environment variable `$CGE_CONFIG_DIR_NAME`, if defined
2. The `.cge` directory, if present under a user's home directory and as defined by the `$HOME` environment variable.
3. The `.ssh` directory, if present under a user's home directory and as defined by the `$HOME` environment variable.

Only keys from the first directory found to contain keys will be used. Enabling verbose mode displays log output, detailing which keys are being used.

5.1.4 CGE Hadoop HDFS Configuration

The CGE CLI requires access to HDFS configuration to retrieve data results and configuration files that may exist there. As such, the value of the `HADOOP_CONF_DIR` environment variable is inspected and relevant configurations files from this directory are used if this variable specifies a valid directory, otherwise the default location `/etc/hadoop/conf` is searched. The system will display log output, which lists configurations files that are used if the verbose mode is enabled.

HDFS and Lustre URL Path Locations

Specify a full URL to the Lustre file system when check-pointing to Lustre. The pathname specified is interpreted relative to the scheme and authority of the data directory URL. To checkpoint to a different scheme, specify the scheme's URL. While check-pointing to Lustre from HDFS, the following path will inform the `checkpoint` command where to store the data:

```
file:/mnt/lustre/my/data/directory
```

- The checkpoint is written exactly as specified by the URL if a full URL is used. This means that an HDFS URL will cause the checkpoint to be written to the path specified in the URL on the HDFS file system described by the rest of the URL, and a file URL (i.e. `file:/path`) will be written to the POSIX file system at the pathname specified in the URL.

- The checkpoint will be written in a directory relative to the data directory used at CGE start up if a relative path (i.e., a simple path with no leading '/' character) is used.
- The pathname will be interpreted within the space specified by the URL of the data directory used at CGE start up if a full pathname but no URL is specified. The checkpoint will be written at the specified path within HDFS if CGE was started using an HDFS URL. The checkpoint will be written at the specified path within the POSIX file space if CGE was started with a simple pathname or file URL.

5.1.5 Cray Graph Engine (CGE) Properties File

A `cge.properties` file can be used to specify some command options, thus eliminating the need to explicitly state them with every command invocation.

The properties file can be:

1. specified via the `--configDir` option
2. specified via the `$CGE_CONFIG_FILE_NAME` environment variable
3. specified via the `$CGE_CONFIG_DIR_NAME` environment variable
4. located in the working directory from which the command-line interface is launched
5. located in the `.cge` directory, which in turn is located under the home directory, as defined by the `$HOME` environment variable

Only the first properties file found will be used. Enabling verbose mode displays output detailing exactly which properties file (if any) are used. If present, values from this file are used unless these are specifically overridden using command line options. Use the `get-configuration` command to view additional detail, such as the locations being searched, which file is used, and the effective properties. The following properties are currently supported:

Table 3. CGE Property Files

Property	Supported Values	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-

Property	Supported Values	Equivalent Command Line Option	Description
			time interactive CLI command each time you start using a server on a new TCP/IP port number combination.
`cge.cli.server.host`	`ServerHost`	`--server-host`	Sets the default host on which the front end launched by the `fe` Command will accept HTTP requests
`cge.cli.server.port`	`cge.cli.server.port`	`--server-port`	Sets the default port number on which the front end launched by the `fe` command will accept HTTP requests
`cge.cli.server.security`	`ShiroConfiguration`	`--security`	Sets the Apache Shiro configuration file used to configure user authentication for the front end.
`cge.cli.server.ssl.enabled`	`True/False`	`--ssl`	Sets whether SSL is enabled for the front end.
`cge.cli.server.ssl.lax`	`True/False`	`--ssl-lax`	Sets whether the SSL configuration for the front end should permit older cyphers and protocols.
`cge.cli.server.ssl.keystore`	`KeystoreFile`	`--keystore`	Sets the location of the Java key store used to provide the SSL certificate for the front en.

Property	Supported Values	Equivalent Command Line Option	Description
`cge.cli.server.ssl.password`	`KeystorePassword`		Sets the password needed to unlock the Java key store which provides the SSL certificate for the front end.
`cge.cli.server.ssl.key-password`	`CertificatePassword`		Sets the password needed to unlock the SSL certificate within the Java keystore.

If there is a properties file that overrides the default value, it will be indicated in the logging and will contain a warning to alert the user of the fact that they have set it in the properties file.

⚠ **CAUTION:** Leaving an out of date properties file around can interfere with correct communications with the database server with no clear reason.

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
```

Restrictions

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

Certain advanced functions 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 the c alias, which invokes the compile command. However, 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
```

⚠ **CAUTION:** A positional parameter that receives no value will be passed through as-is, which will likely result in parser errors.

5.1.6 Create Checkpoints Using the CGE checkpoint Command

The checkpoint command is used to request checkpoint creation. A checkpoint 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. It is a compiled database consisting of a dbQuads, string_table_chars, and string_table_chars.index file.

This command simply accepts a directory path to create the checkpoint in. The checkpoint directory is specified as a URI, which may be a full path, such as file:// or hdfs:///URL. It can also be a relative URI, in which case it will be resolved relative to the base URI on the server, which is the current database directory. If a relative path is used, the path will be evaluated relative to the data directory of the running CGE instance.

By using that directory's path as the checkpoint's path, it is possible to checkpoint to the same data directory the user started from. A successfully created checkpoint will overwrite the existing dbQuads, string_table_chars and string_table_chars.index files, so that the new dataset is retrieved the next time the user starts from that directory. Alternatively, it is also possible to checkpoint to another directory. If the directory already contains a dataset, and the checkpoint succeeds, the dataset will be overwritten.

If the data directory is being moved to a different location, shutdown any instance of CGE that was launched using that data directory before relaunching CGE.

While using the checkpoint command:

- If a full URL is used, the checkpoint is written exactly as specified by the URL, which means that an HDFS URL will cause the checkpoint to be written to the path specified in the URL on the HDFS file system described by the rest of the URL, and a FILE URL (i.e. file:/path) will be written to the POSIX file system at the pathname specified in the URL.
- If a relative path (i.e. a simple path with no leading / character) is used, the checkpoint will be written in a directory relative to the data directory used at CGE start up.
- If a full pathname, but not a URL is specified, the pathname will be interpreted within the space specified by the URL of the data directory used at CGE start up. Therefore, if CGE was started using an HDFS URL, the checkpoint will be written at the specified path within HDFS, otherwise if CGE was started with a simple pathname or FILE URL, the checkpoint will be written at the specified path within the POSIX file space.
- 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.

Examples

Use a relative URL to a file

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

Use a HDFS URL

```
$ cge-cli checkpoint hdfs:///user/db/cp1
```

Use NQuads

If an NQuads file needs to be generated for use with other RDF and SPARQL tools, use the `-q` or `--quads` option of the `checkpoint` command, as shown in the following example:

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

5.1.7 Compile SPARQL Commands Using the CGE `compile` Command

The `compile` command is used to compile SPARQL commands into the logical and/or physical plans that the database server will use for command execution. This can be useful for understanding how the system is interpreting and optimizing a query or update. Specify multiple files to compile a large number of files at the same time.

Compilation Modes

The `-c/--compiler-mode` option is used to specify the desired compilation output type. Supported values include:

Table 4. Compilation Modes

Compilation Mode	Output Mode
`algebra`	The optimized SPARQL algebra for the query/update as 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 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 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. It wil supersede any individual format requests if the `all` option is also specified. The `-a` or `--all` options can also be specified as a shortcut for specifying the `-c all` option.

Compilation Output

By default, the compilation output is sent to standard output and can be redirected to a file if desired. It is recommended to use the `-f` or `--files` option if multiple files need to be processed, or if more than one output type needs to be generated. This will output a file for each input and compilation mode combination in the directory that 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 more information, see the `cge-cli-compile(1)` man page.

Examples

The following example will compile the SPARQL command found in the `example.rq` file into algebraic form and display it to standard output.

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

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 `getTenRows.qr`

```
$ 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 files:

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

5.1.8 Check the Database State Using the CGE echo Command

The echo command checks whether or not the database server is up and able to respond to requests by sending some data to the database server and verifying that the server echoes it back correctly. If the data is sent successfully, the system returns a message saying: `Echoed data received and validated successfully`.

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

Example: Retrieve Database Status

The following command will send the data `Test data` to the server.

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

5.1.9 Launch the CGE Web Server Using the fe Command

The `fe` command launches 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. 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 it is resistant to terminal disconnects.

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. The port used is displayed in the log message. The default port used is 3756. Use the `--server-port` port to specify a different port, if needed, to run the web server on.

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

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

Server Connection Verification

Use the `--ping` option of the `fe` command to verify that the database server is up and running when starting the web server.

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

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

5.1.10 Search Configuration File Locations Using the get-configuration Command

The `get-configuration` command determines the locations of CGE configuration files and the effective properties. This command does not communicate with the database. It inspects the user's local environment and provides information to help understand how configuration is being discovered.

The output of this command includes relevant environment variables, the locations searched for configuration, and whether a file was found. If a file was found, the path to that file is also displayed. Finally, all CGE related properties from that file are listed along with their values, as part of the output.

Example: Search Locations of Configuration Files

```
$ cge-cli get-configuration
/opt/cray/cge/
2.5.1183_r6061c0b_fe2.5.0_20160926.144651_1_2016101912/bin/cge-cli:
line 8: pushd: .: Permission denied
/opt/cray/cge/
2.5.1183_r6061c0b_fe2.5.0_20160926.144651_1_2016101912/bin/cge-cli:
line 11: popd: directory stack empty
0 [main] WARN com.cray.cge.cli.CgeCli  - User data hiding is enabled,
logs will obscure/omit user data.  Set
cge.server.RevealUserDataInLogs=1 in the in-scope cge.properties file
to disable this behaviour.
Environment Variables:
 CGE_CONFIG_FILE_NAME=
 CGE_CONFIG_DIR_NAME=
 HOME=/home/crayusr

Searched Locations:
 1 - /opt/cray/cge/
2.5.1183_r6061c0b_fe2.5.0_20160926.144651_1_2016101912/bin
 2 - /home/crayusr/.cge

Properties File Found? No


Properties
----------
```

5.1.11 Display keyword ID and User Friendly Keyword Name Mappings Using the `keyword-lookup` Command

The `keyword-lookup` command provides the means to lookup mappings between keyword IDs and user-friendly keyword names. These can be used to find the values that need to be passed to the log options when invoking other commands.

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

Examples

Use the `keyword-lookup` command to lookup a specific keyword ID

```
$ cge-cli log-lookup 28
/opt/cray/cge/2.5.1183_r6061c0b_fe2.5.0_20160926.144651_1_2016101912/bin/cge-cli: \
line 8: pushd: .: Permission denied
/opt/cray/cge/2.5.1183_r6061c0b_fe2.5.0_20160926.144651_1_2016101912/bin/cge-cli: \
line 11: popd: directory stack empty
0 [main] WARN com.cray.cge.cli.CgeCli  - User data hiding is enabled, \
logs will obscure/omit user data.  Set cge.server.RevealUserDataInLogs=1 \
in the in-scope cge.properties file to disable this behaviour.
28=ORDR
```

Use the `keyword-lookup` command to lookup a keyword ID.

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

Use the `keyword-lookup` command without any arguments to display the full mapping of levels to names

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

5.1.12 Retrieve Default Server Logging Information Using the `log-info` Command

The `log-info` command retrieves information about the server's default logging configuration. However, the information returned by the `log-info` command does not necessarily reflect the logging settings for individual requests since all commands may use the CGE command-line options to change the log configuration for specific requests.

The server's default log configuration can be used via the `log-reconfigure` command, if needed.

For more information, see `cge-cli-log-info(1)` and `cge-cli-log-reconfigure(1)` man pages.

Examples

In the following example, the text: '`Default Level Info (8)`' indicates that the server is configured with default settings.

```
$ 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}
```

The following example indicates that the server is configured with non-default settings.

```
$ 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)
```

5.1.13 Lookup Mappings Between Log level Values and User Friendly Log Level Names Using the `log-lookup` Command

The `log-lookup` command provides the means to lookup mappings between log level values and user-friendly log level names. These can be used to find the values that need to be passed to the log options, when invoking other commands.

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

Examples

Look up the log level that has a value of 16

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

Look up a level based on the name

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

Retrieve the full mapping of levels to names

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

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

5.1.14 Change the Default Logging Configuration of the CGE Server Using the `log-reconfigure` Command

The `log-reconfigure` command changes the default logging configuration of the server. The information returned by the `log-info` command does not necessarily reflect the logging settings for individual requests since all commands may use the CLI option to change the log configuration for specific requests.

The system will display a message if an incorrect value is specified for the log-level. Upon successful execution of this command, the system returns the message: "`Received success response`".

TIP: 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` and `keyword-lookup` 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 CGE server is running with a large number of images.

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

Example: Change the Default Logging Configuration

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

5.1.15 Retrieve the Default NVP Configurations Using the CGE `nvp-info` Command

The `nvp-info` command retrieves the default server NVP configuration. The information retrieved does not necessarily reflect the NVP settings for individual requests, since commands may change the NVP configuration for specific requests.

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

Example: Retrieve Default NVP Configurations

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

5.1.16 Change Default NVP Configurations Using the CGE `nvp-reconfigure` Command

The `nvp-reconfigure` command modifies the server's default NVP configuration.

Upon successful execution of this command, the system returns a message saying: "`Received success response`". 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 to verify that the changes have taken effect, as shown below:

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

Most of the supported NPVs have a defined range of acceptable values. Values specified outside of those ranges will be normalized 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.

5.1.17 Display Server Output Directory Information Using the `output-info` Command

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

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

Example: Display Server Output Directory Information

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

5.1.18 Change the Server's Output Directory Using the CGE `output-reconfigure` Command

The `output-reconfigure` command modifies the server's output directory that it writes query results to. This directory is specified as a URI. URLs of type `file://` or `hdfs://` may be used. If a relative URI is specified, it will be resolved relative to the base URI of the server, which is the current database directory.

> **TIP:** After executing the `output-reconfigure` command, it is recommended to use the `output-info` 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.

Example: Modify the server's output directory

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

5.1.19 Execute Queries Using the CGE `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 directly specified queries. 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
0   28  1756    0   file:///lus/scratch/rvesse/results/queryResults.
2016-06-13T13.47.22Z000.28889.tsv
```

Here we can see that the database returns a simple tab separated string with the following fields:

Table 6. `query` Command's Output Description

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 display the results in the console:

```
$ cge-cli query --quiet types.rq | cut -d$'\t' -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 the 'Streaming Results' section below.

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. Use the `--stream` option of the `query` command if it is desired to retrieve the final results in an arbitrary format. 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

CAUTION: 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.

Execution of 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
```

A results header is retrieved for each query run.

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

5.1.20 Cray Graph Engine (CGE) Optimizer Configuration

Use the `--opt-off` and `--opt-on` options to perform query optimizer configuration. 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
```

The preceding example will execute the query with the filter placement optimization disabled. The flag will be considered as disabled if both the enabled and disabled flag options are specified. Values of some flags cannot be changed, regardless of the options specified.

CAUTION: Turning optimization off may result in significantly increased memory usage and/or performance degradation. Therefore, it is strongly recommended that the optimizer configuration be changed only when advised to do so by a Cray support engineer.

5.1.21 Shutdown the CGE Server Using the `shutdown` Command

The `shutdown` command instructs the CGE server instance to shut down 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.

This command will not succeed if the server is in a bad state. Standard Linux techniques for killing an application process should be used in this case.

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

Example: Shut down the CGE server

```
$ cge-cli shutdown
```

5.1.22 Execute Sparql Queries and Updates Using 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. Use the `query` command to execute SPARQL queries. Use the `update` command to execute SPARQL updates. Executing queries/updates using the `sparql` command produces the corresponding results for the command. It displays information about the results for queries, whereas it displays a success/failure message as appropriate for updates.

Differences Between the `sparql` and `query` Commands

- 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.

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

Execute all the queries specified in the `commands.txt` file before executing the queries specified in the `extra-command.ru` file

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

5.1.23 Execute Updates on a Database Using the CGE `update` Command

The `update` command executes updates on a database. This command can be used to execute a single update or a sequence of updates. Executing an update returns a message indicating whether the update succeeded or failed.

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. Therefore, updates contained within any specified list files will be executed before any directly specified updates.

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.

Execution of 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. The command will print a success or failure message for each update.

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

Examples

Execute an Update

```
$ 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.

Execute Multiple Updates

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

5.1.24 Create or Inspect a Java Keystore File Using the CGE generate keystore Command

The `generate keystore` command is used to create/inspect a Java keystore file, which is used to enable SSL support for the `fe` command.

This command supports three different modes of operation:

1. Importing an existing SSL certificate
2. Inspecting an existing key store
3. Generating a self signed SSL certificate

Security Concerns

Key store files are protected by passwords so this command may prompt to either enter/create passwords as necessary. As passwords must be entered interactively, this command may fail if run with the `--non-interactive` option. The user will need to know and supply these passwords elsewhere in order for the key store to be used. The related `cge-cli generate properties` command can be used to store the necessary passwords in obfuscated form in the properties file.

Imported Certificates

This is the most frequently used mode. It allows an existing SSL certificate in possession to be imported into a key store file for use by the `fe` command:

```
$ cge-cli generate keystore --importserver.cer
```

This imports the certificate from the `server.cer` file into a key store file in the default location.

⚠ **CAUTION:** In order for the imported certificate to be usable it must contain the private key as well as the Digital signature from the certificate authority. Without the private key a certificate cannot be used for SSL

Key Store Inspection

This mode can be used to inspect an existing key store to see what certificate is present in it. For example:

```
$ cge-cli generate keystore --display
```

Self-signed Certificate Generation

⚠ **CAUTION:** This mode should only be used for testing purposes. Using a self-signed certificate in a production environment is insecure and not recommended.

In this mode a self-signed certificate is generated and added to the key store. This can be used to test the use of SSL without the need to first obtain a certificate from a recognised certificate authority. However the certificates generated in this way are inherently insecure, may not be trusted by many other tools and should be avoided wherever possible.

```
$ cge-cli generate keystore --self-signed
```

This will prompt the user to enter a variety of identifying information for their certificate, and adds the resulting certificate to the key store ready for use.

5.1.25 Generate a Shiro Configuration Template Using the `generate shiro` Command

The `generate shiro` command is part of the `cge-cli generate` command group and generates a Shiro configuration template that can be customized as desired. It enables users to quickly create a configuration that can be used with the `fe` command to provide user authentication.

Available templates

The following templates are available through the `generate shiro` command:

Table 8. `generate shiro` Templates

Template	Description
`ldap`	A template that can be customised to allow integration with a LDAP server, i.e. it allows authentication to be deferred to an existing LDAP service
`forms`	A template that has both locally defined user accounts and roles, it uses forms authentication
`simple`	A template that has locally defined user accounts and uses HTTP Basic authentication
`anon`	A template that enables anonymous access, i.e. no user authentication

Example: Generate a Shiro Configuration

The following example will generate a Shiro configuration based upon the LDAP template to standard output. The configuration is redirected to the `example.ini` file, where it can be edited as needed.

```
$ cge-cli generate shiro ldap > example.ini
```

5.1.26 Create a Properties File Using the CGE `generate properties` Command

The `generate properties` command is part of the `cge-cli generate` command group and helps create a properties file that can be used to provide a variety of configuration to commands, without needing to specify it directly at the command-line. This command can either create/modify a properties file, so it can be used to create an entirely new configuration, or use it to update an existing configuration.

The options supplied to this command are simply added/updated in the relevant properties file, instead of being used for their normal function. Additionally, there are some options specific to this command that control which properties file is created/modified, and whether modifications are merged with, or if they overwrite existing properties in that file.

The default behaviour of this command is to modify existing properties. The returned properties file is the result of reading the existing properties and applying any modifications requested by this command. If it is preferred to create an entirely new set of properties, use the `--overwrite` option to specify that existing properties are not preserved. It is generally best to be explicit about which properties file needs to be modified using the `-f` or `--file` options, otherwise an incorrect properties file ma be modified. The logging output of this command will explicitly note which properties file is being modified.

Setting values in the properties file does not guarantee that they will be used. Any property which can also be set via a command-line option can be overridden by specifying that option. The logging output will indicate when a property has been used and when a property has been overridden by a command line option.

Basic Usage

The following example generates a properties file in one of the default locations that `cge-cli` will search for it:

```
$ cge-cli generate properties -f ~/.cge/cge.properties --db-port 1234
```

Advanced Usage

The following example overwrites an existing properties file and specifies several properties, including one that does not have a specific command-line option to set it:

```
$ cge-cli generate properties -f ~/.cge/cge.properties --overwrite \
--db-host example.mycompany.com --db-port 1234 -p cge.server.RevealUserDataInLogs 1 --ssl-passwords
```

Table 9. Command specific options

Option	Value(s)	Example Usage	Description
`-f` `--file`	`PropertiesFile`	`-f ~/.cge/cge.properties`	Provides the path to the properties file that needs to be created/modified.
`--overwrite`		`--overwrite`	When set, indicates that any existing properties file at specified/automatically discovered location should be overwritten. The default behaviour is to first read in the file if it exists meaning that any existing properties not being modified by this command are left intact. If you specify this option any existing properties are lost.
`-p` `--property`	`Key Value`	`-p cge.server.RevealUserDataInLogs 1`	When set, indicates that the given property and value should be added to the properties file. This can be used to add any property which does not have a specific option for modifying it.
`--ssl-passwords`		`--ssl-passwords`	When set, will prompt for passwords used to secure the Java key store which contains the SSL certificate use by the `cge-cli fe` command These passwords will be stored in the properties file in obfuscated form to provide some protection from casual inspection. You should apply appropriate

Option	Value(s)	Example Usage	Description
			permissions to the properties file to fully protect these.

5.2 CGE GUI

CGE provides a simple interface for access via a 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.

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.

⚠ **CAUTION:** 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.

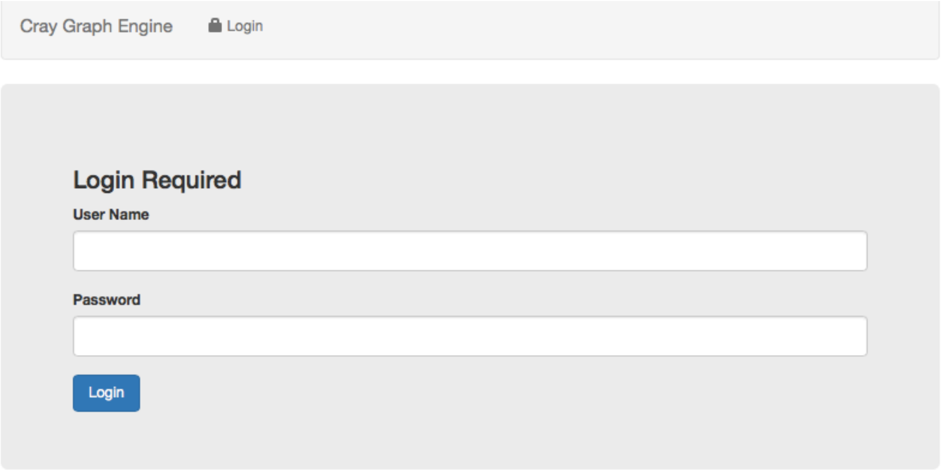
The results format received in the browser is dictated by the HTTP Accept header that your browser sends (or conversely that your programmatic HTTP client sends). The '`Force text/plain as the response Content`' option controls the Content-Type header that the front end responds with, which affects how the browser interprets the response. Depending on the browser if this option is disabled (the default) then this might mean that it downloads/offers to save the response to a file rather than displaying it in the browser, enabling the aforementioned option changes the response Content-Type to always be text/plain regardless of what format the front end actually outputs which forces the browser to display the response in the browser itself. If it is needed to display the results in a different format, customise the HTTP Accept header accordingly, most browsers have some means to configure this. For example in Firefox navigate to **About**>**Config**. Click through the warning if it appears and then search for accept and edit the value of the `network.http.accept.default` setting to add the desired content types. The closest thing to plain text that the front end will produce is text/tab-separated-values. Most browsers include `application/xml` in their default accept header, which mean you will typically get SPARQL XML results by default (or RDF/XML if it were a `CONSTRUCT` query).

Logging on to the CGE UI

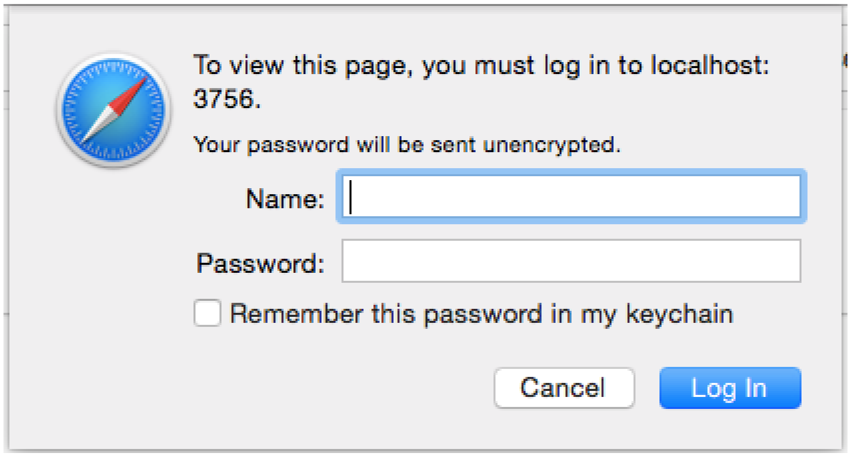
The CGE UI can then be accessed by pointing the browser at: `http://localhost:3756/dataset/`.

If you have configured the server to perform user authentication, the first thing you will see is one of the following screens, depending on what authentication method has been configured. For more information, see *CGE Security* on page 81.

- When configured for forms authentication you will see you the following screen:



When configured for basic authentication, the browser will prompt for credentials like so:



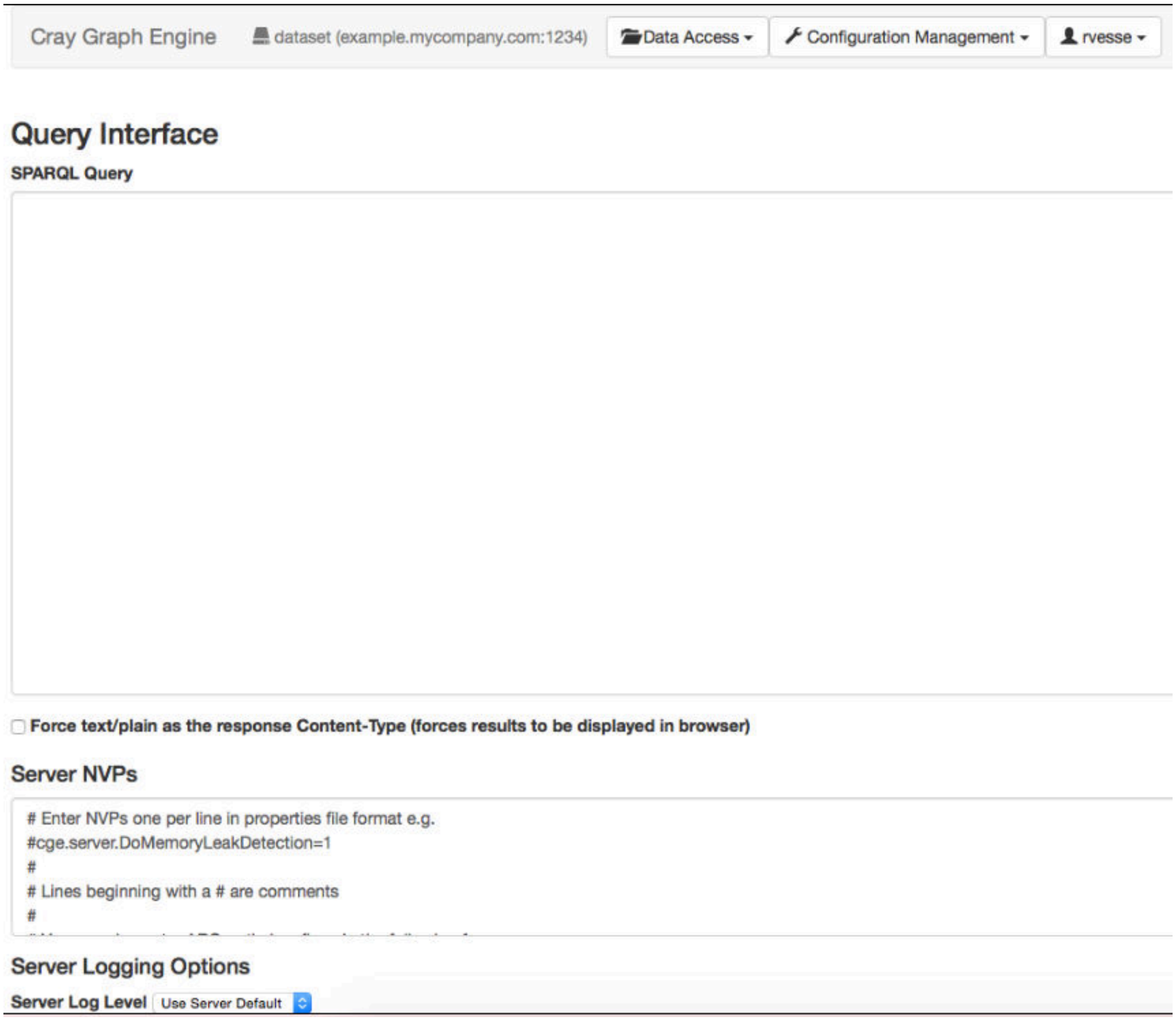
To view this page, you must log in to localhost: 3756.

Your password will be sent unencrypted.

The exact format of this dialogue will depend upon the browser you're using, this example is from Safari. Either way the user will need to enter their credentials in order to log in.

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

Figure 2. Cray Graph Engine User Interface


Cray Graph Engine
dataset (example.mycompany.com:1234)
Data Access ▾
Configuration Management ▾
rvesse ▾

Query Interface

SPARQL Query

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

Server NVPs

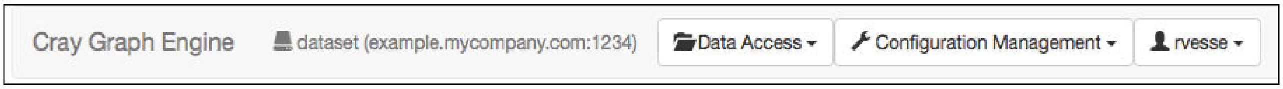
```
# Enter NVPs one per line in properties file format e.g.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

Server Logging Options

Server Log Level Use Server Default

At the top of the page you will find the navigation bar:

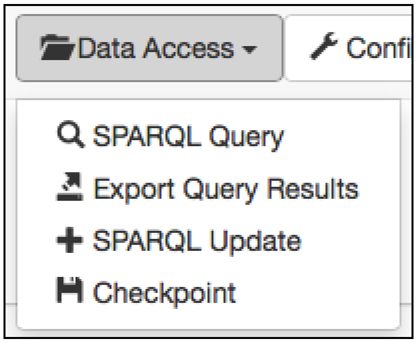
Figure 3. CGE UI Navigation Bar



This provides a number of useful pieces of information. Firstly it indicates the underlying database server that the front end will be connecting to. In this example the underlying database server is on *example.mycompany*.com: 1234

There are then three menus which provide access to the various functionalities of the server. The data access menu contains the following:

Figure 4. Data Access Menu Options

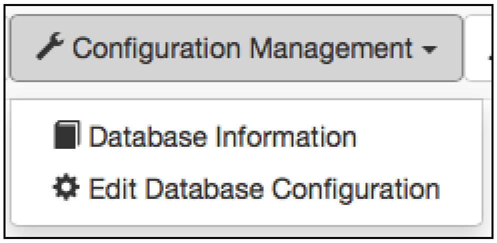


The options in the menu include:

- **SPARQL Query** enables making queries
- **Export Query Results** allows you to make a query but only returns meta data about where the results have been saved to disk
- **SPARQL Update** enables making updates
- **Checkpoint** enables checkpointing the database to disk

The configurations management menu contains the following options:

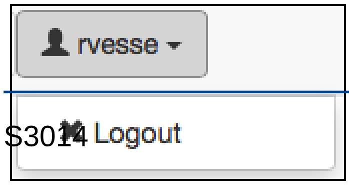
Figure 5. Configuration Management Menu Options



- **Database Information** provides access to the current configuration of the server
- **Edit Database Configuration** allows you to edit that configuration

Finally the user menu shows the currently logged in username and provides access to logout functionality:

Figure 6. User Menu options



If you have not configured user authentication, the system will instead display the following warning:

Figure 7. Insecure Mode Warning

5.2.1 Launch the CGE 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 *CGE CLI*.

5.2.2 Execute SPARQL Queries Using the CGE UI

About this task

The Cray Graph Engine (CGE) **Query Interface** allows executing SPARQL queries on a loaded RDF database running within CGE. The main feature of this interface is the text field for entering queries to execute. Secondly, there is a check box that enables specifies that the server returns the query results with a

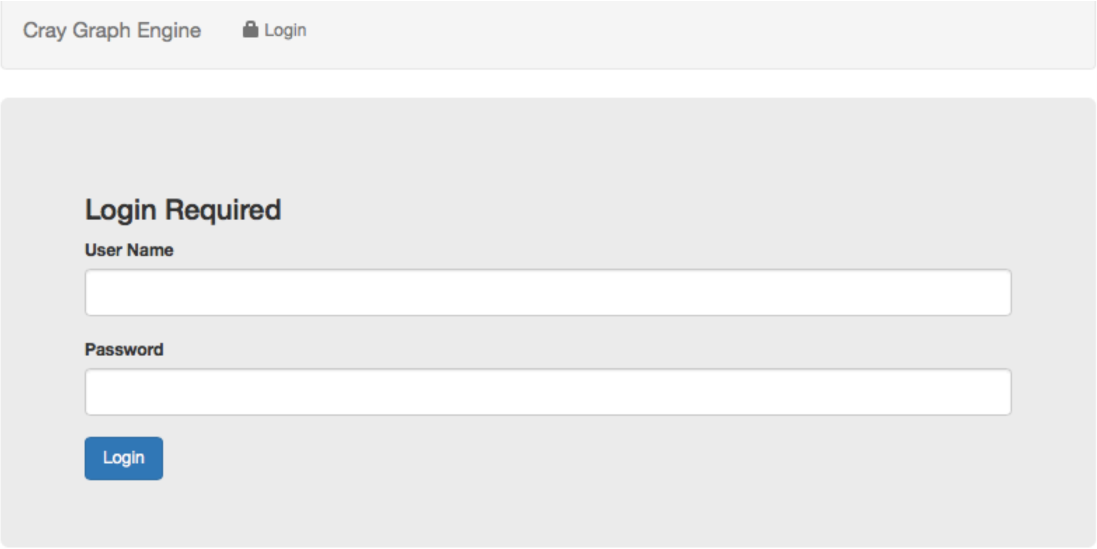
`Content-Type` header value of `text/plain`, which will force the browser to display the results as many browsers will download the results rather than display them by default. The rest of the options seen in this interface are described later in the **Advanced Options** section.

The browser interface uses standard HTTP content negotiation to determine the format in which to return the query results, most browsers out of the box will receive results in an `XML/JSON` format:

Procedure

1. Optional: Log on to the CGE UI by pointing a browser at `http://machine-login1:3756/login`, entering credentials and then selecting the **Login** button.

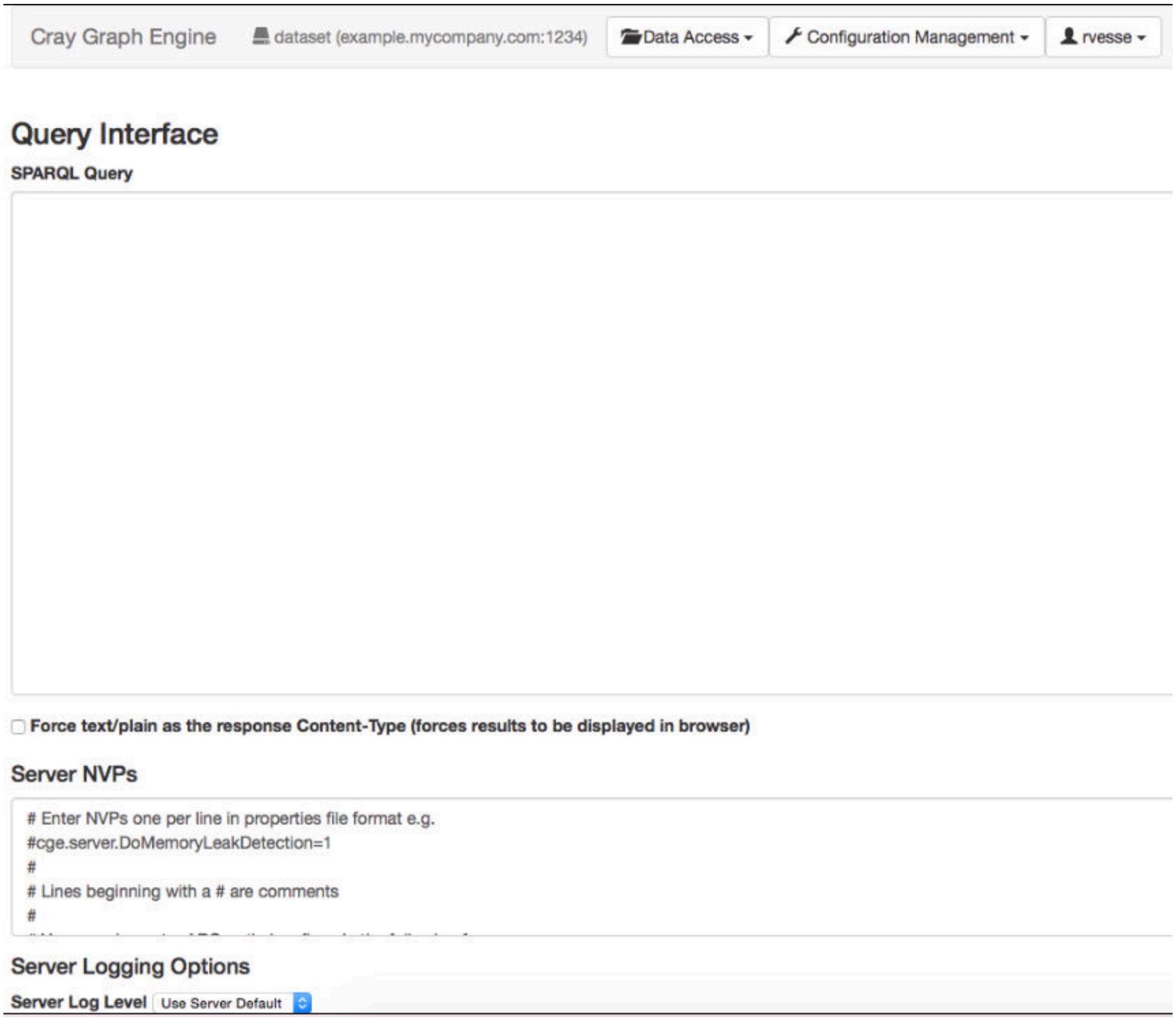
   This step is optional, depending on how the interface has been configured



2. Access the CGE **Query Interface** using one of the following mechanisms:

- Point the browser at `http://machine:3756/dataset/query`
- Select the **Query Interface** link from the **Data Access** drop down on the CGE **Query Interface** UI.

Figure 8. Query Interface



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

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

3. Execute a SPARQL query,by entering 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.

4. 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.

5.2.2.1 Get Query Metadata

Sometimes it may not be desired to get all the results delivered over HTTP. Instead, it may be needed to simply submit a query whose results will be processed later. To do this, use the export query results endpoint accessed at: http://*`machine`*:3756/dataset/export-results, where *`machine`* is used as an example for the machine name.

This interface is functionally identical to the **Query** interface. The endpoints differ only in the format of the response. The export results endpoint return only the meta data about query results. This is similar to the default behaviour of the `query` command. The meta data is returned in one of three formats, where the response format to use is determined by content negotiation.

Table 10. Query Metadata

Format	Example Response	Content Types
Tab separated values (TSV)	`0 100 0 2 /tmp/results.tsv`	● `text/plain` ● `text/tab-separated-values`
XML	`<?xml version="1.0" encoding="UTF-8"?> <cge-results> <query><![CDATA[SELECT * WHERE { } ]]></query> <count>100</count> <size>0</size> <time>2</time> <status>0</status> <location>/tmp/results.tsv</location> </cge-results>`	`application/xml`
JSON	`{ "query" : "SELECT * \nWHERE\n { }\n" , "count" : 100 , "size" : 0 , "time" : 2 , "status" : 0 , "location" : "/tmp/results.tsv" }`	`application/json`

This interface only supports `SELECT` queries. Any other queries will be rejected, this is because the meta data is only accurate and complete for `SELECT` queries.

5.2.3 Execute SPARQL Updates Using the CGE Update Interface

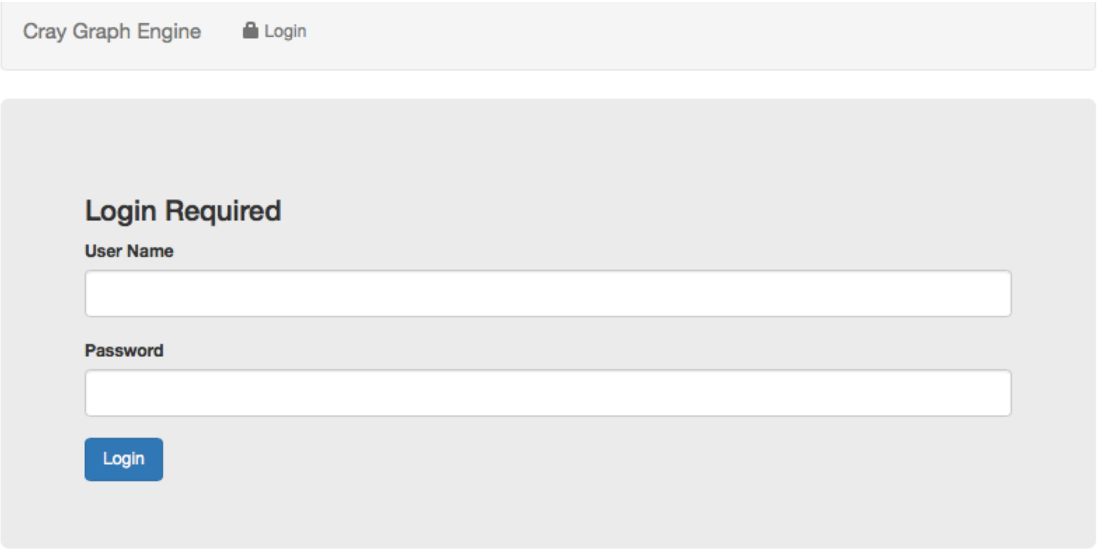
About this task

The Cray Graph Engine (CGE) **Update Interface** enables executing 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. Use the CGE **Update 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

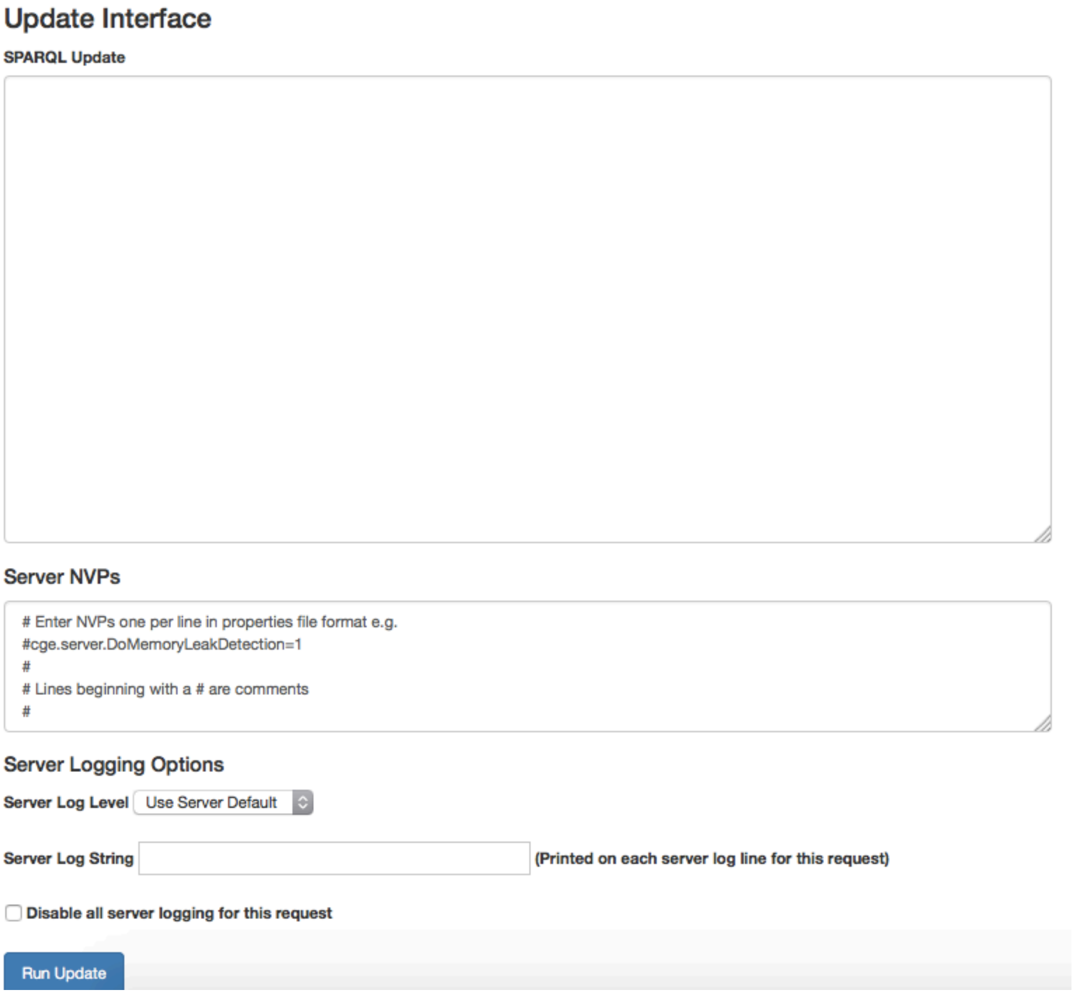
1. Optional: Log on to the CGE UI by pointing a browser at `http://machine-login1:3756/login`, entering credentials and then selecting the **Login** button.

   This step is optional, depending on how the interface has been configured.



2. Access CGE's **Update Interface** by selecting one of the following mechanism:
   - Point a browser at `http://machine:3756/dataset/update`
   - Select **Sparql Update** from the **Data Access** drop down on the CGE UI.

Figure 9. CGE Update Interface



Server NVPs:

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

Server Log Level: Use Server Default

□ Disable all server logging for this request

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

4. 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.

5.2.4 Create a Checkpoint Using the CGE UI

About this task

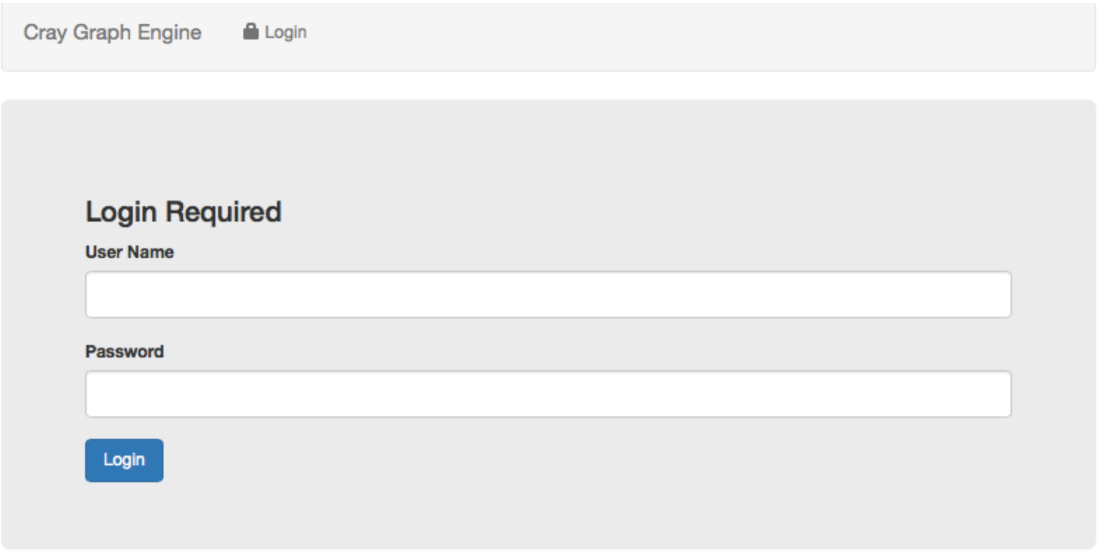
When a database is started for the first time its initial state is considered to be a checkpoint. When a change is made to the database, its state can be preserved by creating a checkpoint. This preserves a copy of the previous in-memory database. Creating a checkpoint creates a persistent record of 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. Optional: Log on to the CGE UI by pointing a browser at `http://machine-login1:3756/login`, entering credentials and then selecting the **Login** button.

   This step is optional, depending on how the interface has been configured

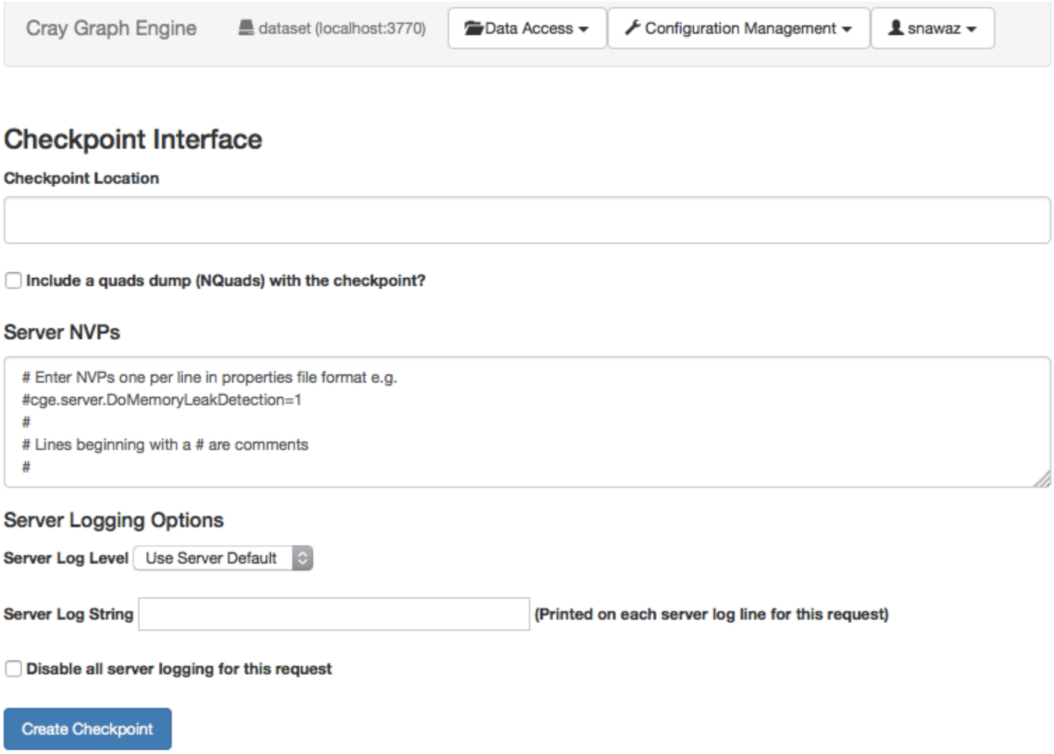
   

2. Access the **Checkpoint Interface** using one of the following mechanisms:

   - Point the browser at `http://machine:3756/dataset/checkpoint`, where machine is the machine running CGE's web server.
   - Select **Checkpoint** from the **Data Access** drop down.

   This brings up the **Checkpoint Interface**, as shown below:

Figure 10. Creating a Checkpoint



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.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

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

3. 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 read-only.

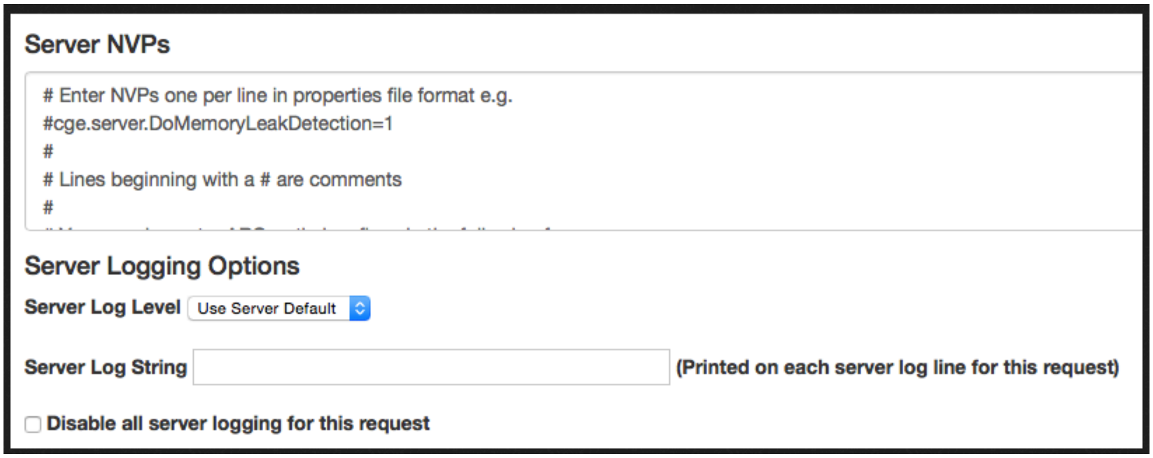
4. Select the **Create Checkpoint** button to create the checkpoint. This will return a success/failure message as appropriate, as shown in the following example output:

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

5.2.5 Cray Graph Engine (CGE) Advanced Options

CGE provides a number of advanced options that can be used to change the behavior of the database server for a specific request. Some of these options impact the server, whereas others impact individual requests. To access this interface, select **Edit Database Configuration** from the **Data Access** drop down. The user interface for configuring advanced options is shown in the following figure:

Figure 11. Server Name Value Pairs



Server NVPs

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

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

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 '*Edit Server Configurations Using the CGE UI*' topic of this publication.

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 behavior 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.

5.2.6 View Server Configurations Using the CGE UI

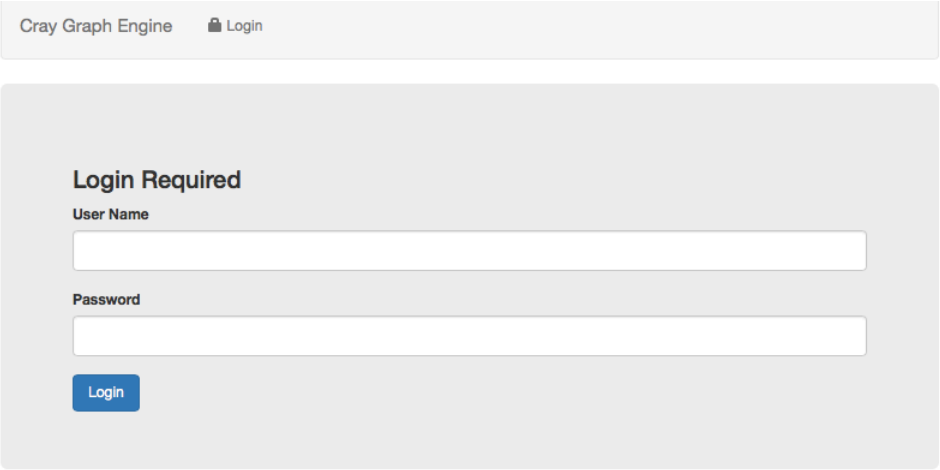
About this task

The **Server Information** interface enables viewing all the server configuration settings defined in the system.

Procedure

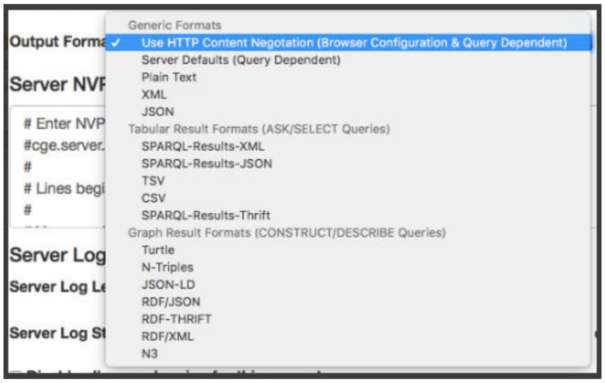
1. Optional: Log on to the CGE UI by pointing a browser at `http://machine-login1:3756/login`, entering credentials and then selecting the **Login** button.

   This step is optional, depending on how the interface has been configured



2. Access the **Database Information** interface using one of the following mechanisms:
   - Point a browser at `http://`*`machine`*`:3756/dataset/info`, where *`machine`* is the machine running the CGE web server.
   - Select **Database Information** from the **Configuration Management** drop down on the CGE UI

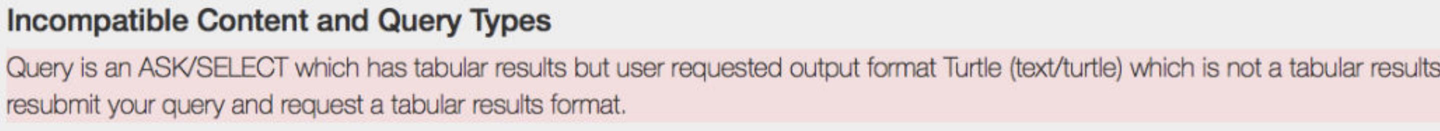
Figure 12. Server Configurations



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.

Here you can select how you would like to receive your query results. The **Generic Formats** section at the top of the list enables specifying the general form of the desired output with the server automatically selecting a suitable output format depending on the type of query being made. It is also possible to select a specific tabular or graph results format from those supported. Note that if a specific format is selected, it must be compatible with the type of query being submitted, otherwise the system will return the following error:

Figure 13. Error Example



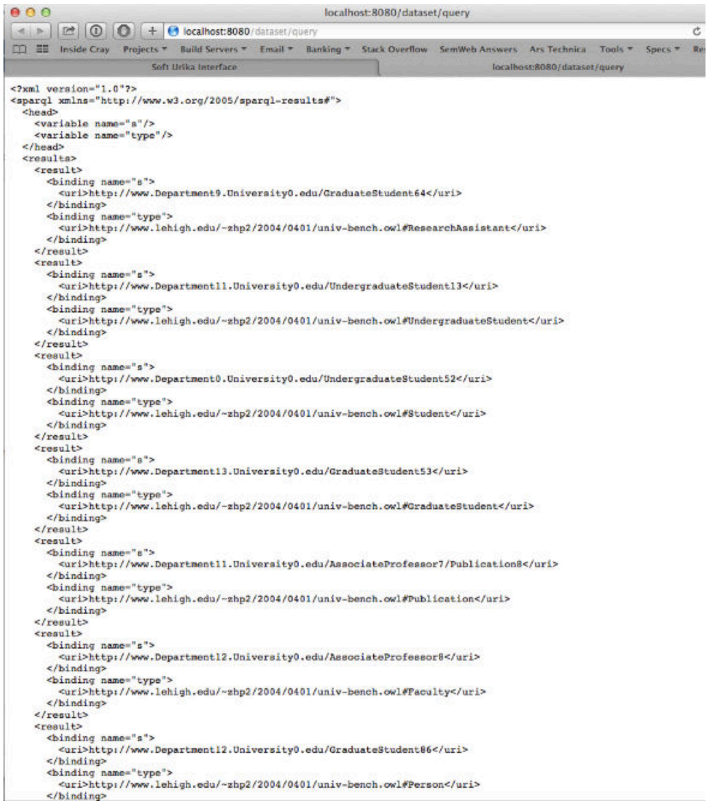
Incompatible Content and Query Types

Query is an ASK/SELECT which has tabular results but user requested output format Turtle (text/turtle) which is not a tabular results
resubmit your query and request a tabular results format.

If the system returns this error, select an alternative output format that is compatible with the type of query being executed.

To run the query simply, select the **Run Query** button, which will send the query to the server and then deliver the results to the browser. The browser interface uses the selected **Output Format** above or standard HTTP content negotiation to determine the output format in which to return the query results. Most browsers out of the box will receive results in an XML/JSON format:

Figure 14. JSON format Example



```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="s"/>
    <variable name="type"/>
  </head>
  <results>
    <result>
      <binding name="s">
        <uri>http://www.Department9.University0.edu/GraduateStudent64</uri>
      </binding>
      <binding name="type">
        <uri>http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#ResearchAssistant</uri>
      </binding>
    </result>
    <result>
      <binding name="s">
        <uri>http://www.Department11.University0.edu/UndergraduateStudent13</uri>
      </binding>
      <binding name="type">
        <uri>http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#UndergraduateStudent</uri>
      </binding>
    </result>
    <result>
      <binding name="s">
        <uri>http://www.Department0.University0.edu/UndergraduateStudent52</uri>
      </binding>
      <binding name="type">
        <uri>http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Student</uri>
      </binding>
    </result>
    <result>
      <binding name="s">
        <uri>http://www.Department13.University0.edu/GraduateStudent53</uri>
      </binding>
      <binding name="type">
        <uri>http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#GraduateStudent</uri>
      </binding>
    </result>
    <result>
      <binding name="s">
        <uri>http://www.Department11.University0.edu/AssociateProfessor7/Publication8</uri>
      </binding>
      <binding name="type">
        <uri>http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Publication</uri>
      </binding>
    </result>
    <result>
      <binding name="s">
        <uri>http://www.Department12.University0.edu/AssociateProfessor8</uri>
      </binding>
      <binding name="type">
        <uri>http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Faculty</uri>
      </binding>
    </result>
    <result>
      <binding name="s">
        <uri>http://www.Department12.University0.edu/GraduateStudent86</uri>
      </binding>
      <binding name="type">
        <uri>http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Person</uri>
      </binding>
```

5.2.7 Edit Server Configurations Using the CGE UI

About this task

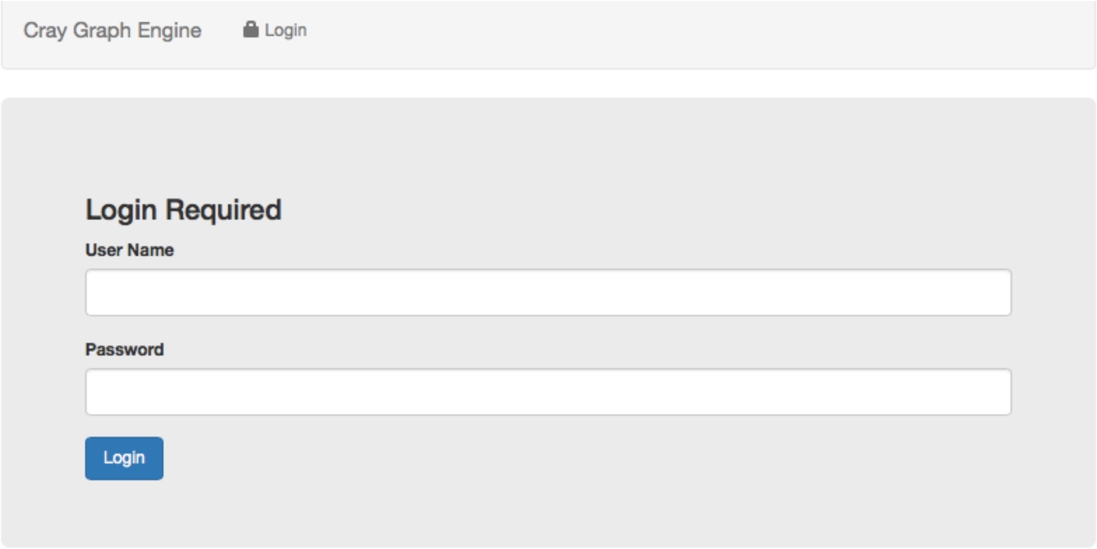
The **Edit Server Configuration** interface allows editing server configurations.

⚠ **CAUTION:** Modifying server configuration settings can adversely affect performance, especially if it is changed to point to a relatively slow file system. Therefore, it is recommended not to change server configuration settings, unless specifically instructed to do so by a Cray representative in order to gather information for diagnosing issues.

Procedure

1. Optional: Log on to the CGE UI by pointing a browser at `http://machine-login1:3756/login`, entering credentials and then selecting the **Login** button.

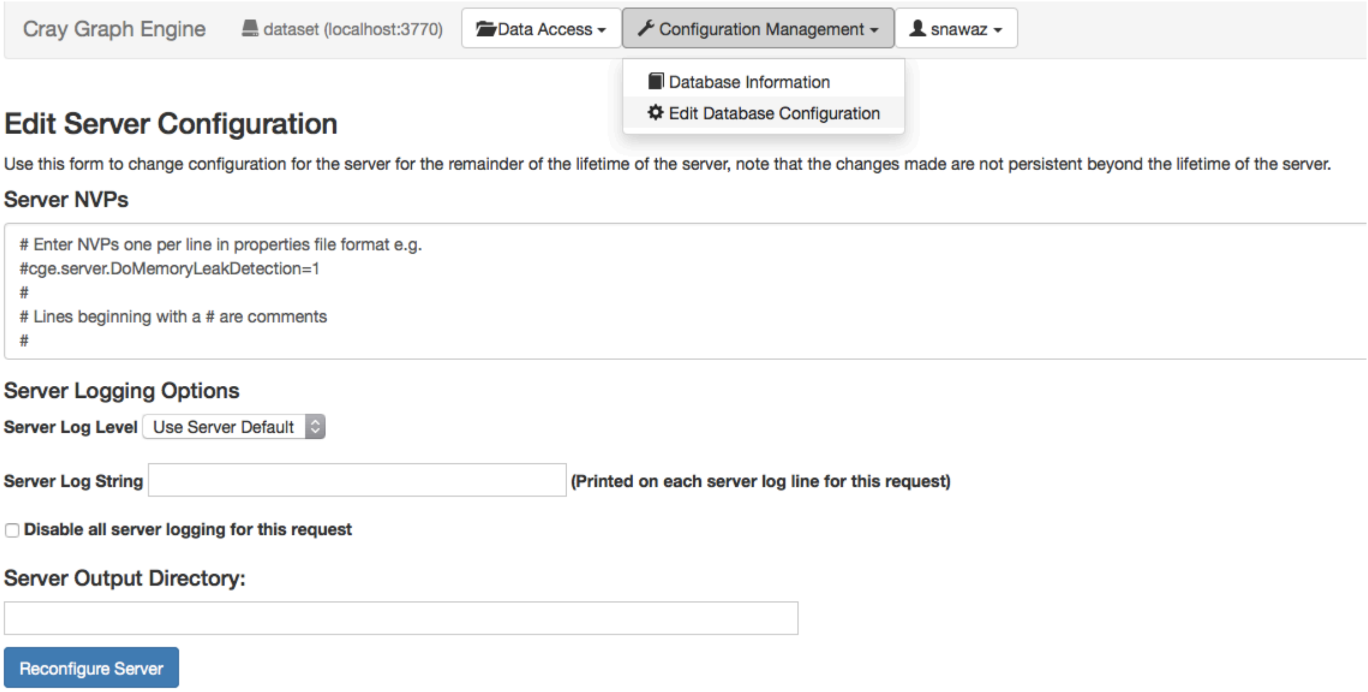
   This step is optional, depending on how the interface has been configured

2. Access the CGE **Edit Server Configuration** interface, by using one of the following mechanisms:

   - Point a browser at `http://`*`machine`*`:3756/dataset/config`, where *`machine`* is the machine running CGE's web server.
   - Select **Edit Database Configuration** from the **Configuration Management** drop down on the CGE UI.

Figure 15. Editing Server Configurations



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.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

Server Logging Options

(Printed on each server log line for this request)

□ Disable all server logging for this request

Server Output Directory:

3. Select the desired server 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.

4. Select the **Reconfigure Server** button when the changes have been made.

   Unlike the options presented in the other interfaces, the values set from this interface persist for the lifetime of the server and become the new defaults.

   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
   ```

5.2.8 Control Options

In most cases it will not be needed to change server configuration settings, unless a Cray support representative specifically requests, in order to gather information for diagnosing issues. However, there are some 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 11. 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`
`cge.server.BuddyMaxGBs`	Sets the upper limit on the amount of memory used by the big buddy allocator. The value of `BuddyMaxGBs` must be a non-negative integer value and is used to specify the maximum number of gigabytes allocated for the big buddy allocator. For example, setting the value to 50 will set the upper limit on the memory allocated for the big buddy allocator to 50 GB. By default, the limit is set to 128 GB and the maximum is 1 TB. Setting this parameter to 0 will disable the limit.	`128 GB`
`cge.server.LittleBuddyMaxGBs`	Sets the upper limit on the amount of memory used by the little buddy allocator. The value of `LittleBuddyMaxGBs` must be a non-negative integer value and is used to specify the maximum number of	`16 GB`

Parameter	Description	Default Value
	Gigabytes allocated for the little buddy allocator. For example, setting the value to 8 will set the upper limit on the memory allocated for the little buddy allocator to 8 GB. By default, the limit is set to 16 GB and the maximum is 128 GB. Setting this parameter to 0 will disable the limit.	
`cge.server.RevealUserDataInLogs`	Specifies whether or not to obscure user data output to logs. If log data is obscured for the given application run, CGE issues the warning: "`User data obscurred. set cge.server.RevealUserDataInLogs=1 to show`". Setting the value of this parameter to `1` informs CGE to not obscure user data output to logs.	By default, obscures user data that is output to the logs.
`cge.server.BuddyMemPercent`	Set the percentage of node memory used for the large persistent allocators.	35
`cge.server.PersistBuddyMemPercent`	set the percentage of node memory used for the large non-persistent allocators.	25
NVPs for GraphML Support		
`cge.server.ExportGMLRDFEnable`	Setting this NVP to `1` will cause CGE to export the quads generated for a given GraphML file to an `nt` file of the same name as the input GraphML file but with the `nt` extension	Off
`cge.server.GMLInsertPrefix`	Setting this to `1` will cause CGE to insert the `urn:` prefix when converting identifiers for graphs, nodes, and edges to URIs.	On
`cge.server.GMLCheckPrefix`	Setting this to `1` will cause CGE to check an identifier for a known prefix before inserting the urn: default prefix.	CGE inserts the `urn:` prefix by default.
`cge.server.BCmaxActiveLevels`	Used for handling graphs of large diameter while using the Betweenness Centrality graph algorithm.	100

5.3 SPARQL Endpoints

CGE provides standards compliant SPARQL endpoints via the `cge-cli fe command`. When run this command launches an embedded Jetty web server that provides SPARQL 1.1 protocol compliant endpoints that may be used by any SPARQL aware tools to make queries and updates against CGE.

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

Web Server

The web server is a standard Java servlets based web application, for ease of deployment and usage we host this in an embedded Jetty server. The web application consists of a bunch of Java servlets defined in the `cge-sparql-server` module with one for each service provided by the CGE SPARQL server. Additionally there is some static HTML content, each piece of HTML content actually represents only a small portion of a page of the browser interface. These pieces are served and combined dynamically by a simple templating engine, this allows for easily tweaking portions of the browser interface and having those be automatically reflected on all pages of the interface.

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 12. 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.

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.

> **NOTE:** The results format received in the browser is dictated by the HTTP Accept header that your browser sends (or conversely that your programmatic HTTP client sends). The '`Force text/plain as the response Content`' option controls the `Content-Type` header that the front end responds with, which affects how the browser interprets the response. Depending on the browser if this option is disabled (the default) then this might mean that it downloads/offers to save the response to a file rather than displaying it in the browser, enabling the aforementioned option changes the response `Content-Type` to always be text/plain regardless of what format the front end actually outputs which forces the browser to display the response in the browser itself. If it is needed to display the results in a different format, customise the HTTP Accept header accordingly, most browsers have some means to configure this. For example in Firefox navigate to **About**>**Config**. Click through the warning if it appears and then search for accept and edit the value of the `network.http.accept.default` setting to add the desired content types. The closest thing to plain text that the front end will produce is text/tab-separated-values. Most browsers include `application/xml` in their default accept header, which mean you will typically get SPARQL XML results by default (or RDF/XML if it were a `CONSTRUCT` query).

The following standard formats are supported by the query endpoint:

Table 13. 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 ● SPARQL Results Thrift
CONSTRUCT and DESCRIBE	● NTriples ● Turtle ● RDF/XML ● RDF/JSON ● RDF/Thrift ● JSON-LD

Standard HTTP behavior 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 14. 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 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

Parameter	Example	Purpose
		● `4 = Warn` ● `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

`output`	`output=json`
	OR
	`output=text/turtle`
		Used to specify the desired output format without needing to modify the `Accept` header. It allows forcing a specific output format to be used provided it is compatible with the query being submitted.
		It accepts `default`, `XML`, `text`, and `json`, which will select the servers preferred default output or a suitable XML, plain text or JSON representation for the query type. Any other value is treated as a MIME type for the output format, only MIME types that the server knows how to return can be used. The list of acceptable values is the list of MIME types associated with the aforementioned supported output formats.

5.4 Create and Use a Database

Prerequisites

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

About this task

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

Procedure

1. If the data is not in RDF format, convert the data to RDF.

2. If the 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 the data resides in more than one file, create a `graph.info` file and add the names of the RDF file to that file.

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. *`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`

   Executing multiple update commands at a time is not supported currently. Updates should be split into separate files and/or submissions.Collectively, the aforementioned files are the disk representation of the binary version of the database which can be reloaded into CGE. When the CGE application is launched again and the same database directory is specified, 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 the data. This is because inferencing is turned on by default. It can 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 &
   ```

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

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

      1. Connect to the CGE UI by pointing the 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 respectively. Optionally, server configuration parameters can also be specified to control the query/update.

   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.

6 Query Cancellation

The CGE Server will cancel a request any time the client making the request disconnects from the server, or if the request exceeds an NVP configurable timeout value. Request cancellation can occur between operations within a query, inside the merge operation, inside the filter operation or inside the `group-by` operation. The first two of these will always recognize request cancellation, while cancellation must be explicitly enabled for the filter and group-by operations. Some of this optimization is disabled when cancellation is enabled, resting in slower operation. Set the `server.LoopInterruptGranularitySeconds` NVP value to a non-zero value (1 is a good choice) to enable cancellation in filter and group-by operations. This value can be set either in the `cge.properties` file or in the NVPs sent with a specific query. The maximum number of seconds defaults to 1 in merge operations, but can be increased by increasing this setting.

Wait for the memory allocation process to complete if query cancellation is taking longer than several minutes. Restart the CGE server on additional nodes to provide additional memory, thus preventing queries from slowing down frequently.

Process and Request Termination

The CGE CLI acts as a client to the database server. When a command requiring a connection to the database is executed, the control flow is as follows:

1. Command performs any client side validation and processing that is necessary for the requested action
2. A request to the database is prepared
3. A connection to the database is established
4. The request is submitted to the database
5. The client waits until it receives a response from the database
6. The response is processed as necessary
7. Command returns results, if any, and exits with an appropriate exit code or continues on to the next requested action

If the process is terminated during steps four and five, CGE will make a best effort to terminate the submitted request by forcibly disconnecting the active connection. The database server will spot the disconnection and will terminate request processing at the next cancellation point. Cancellation may not be immediate and may take a long time to occur, depending on the current operation. When running the CGE SPARQL server, use the active connections interface to explicitly cancel requests submitted via HTTP.

Therefore after submitting a cancellation request for terminating a long running query, it may not be possible to submit further requests until the database has either cancelled/completed the previous request. Typically when this happens the system will return an error stating that the command line timed out trying to connect to the database. If query cancellation takes more than several minutes to complete, restart the CGE server on a larger block of nodes to provide additional memory and prevent queries from slowing down due to lack of memory. Restarting the database will lose any in-memory changes that were not yet checkpointed to disk. For databases with read/write workloads, checkpoint regularly prior to executing long running queries.

Query Cancellation Using a Timeout

Setting the `server.QueryTimeout` NVP value while submitting a query is another way of cancelling long running queries. The query will time out when the number of specified is reached, causing it to fail and send back a failure message. This can be useful when developing queries or when the duration of execution is unknown. Configure this setting either in the `cge.properties` file or specify it with the submitted query.

NVPs Associated with Query Cancellation

- `server.QueryTimeout` - Set a timeout value in seconds for a given query or all queries
- `server.LoopInterruptGranularitySeconds` - When non-zero, enables cancellation in the filter operation. When greater than 1 increases the interval, at which cancellation will be checked in merge and filter operations.

In addition to these user NVPs, there are three NVPs provided for internal testing purposes. These are listed here because setting them will cause a dramatic performance degradation for queries.

⚠ **WARNING:** The default value for NVPs is 0. Do not modify this value unless advised by Cray Support for debugging purposes.

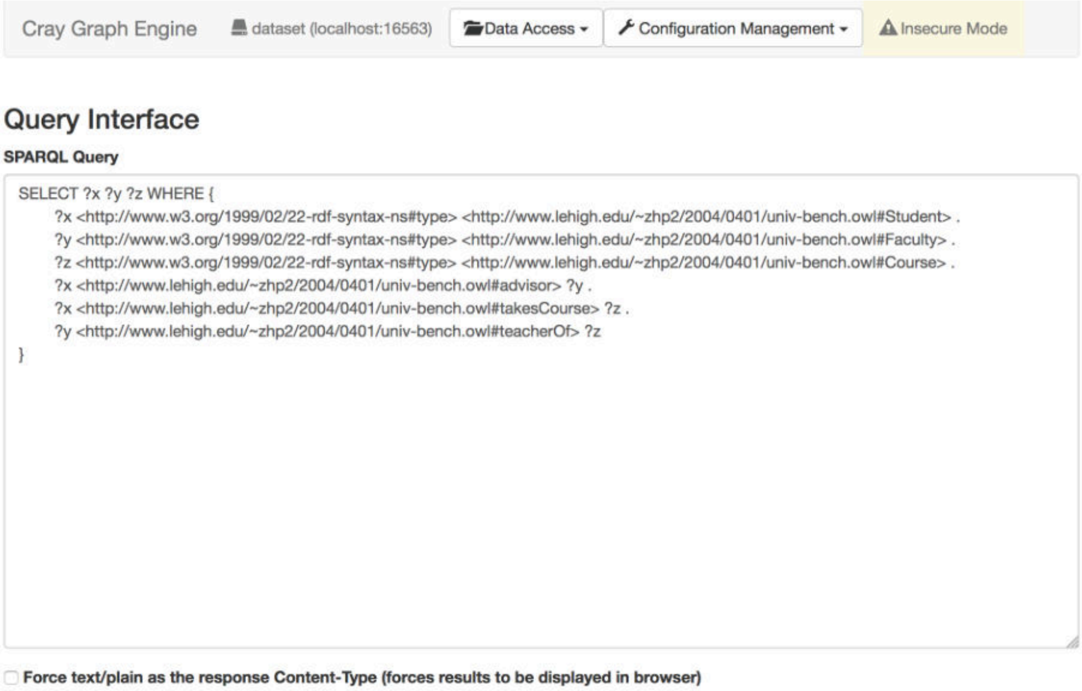
- `server.TestCancellationDispatcherPauseSeconds`
- `server.TestCancellationFilterPauseSeconds`
- `server.TestCancellationMakemergePauseSeconds`
- `server.TestCancellationGroupInitHurisPauseSeconds`
- `server.TestCancellationGroupEvalArgPauseSeconds`

6.1 Cancel a Query Using the CGE Web UI

About this task

If a user has submitted a query using the CGE web UI (which is launched via the `cge-cli fe` command), the sytsem will present a web-browser similar to the following:

Figure 16. CGE Query Submitted Using the UI



```
SELECT ?x ?y ?z WHERE {
      ?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Student> .
      ?y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Faculty> .
      ?z <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Course> .
      ?x <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#advisor> ?y .
      ?x <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse> ?z .
      ?y <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#teacherOf> ?z
}
```

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

Procedure

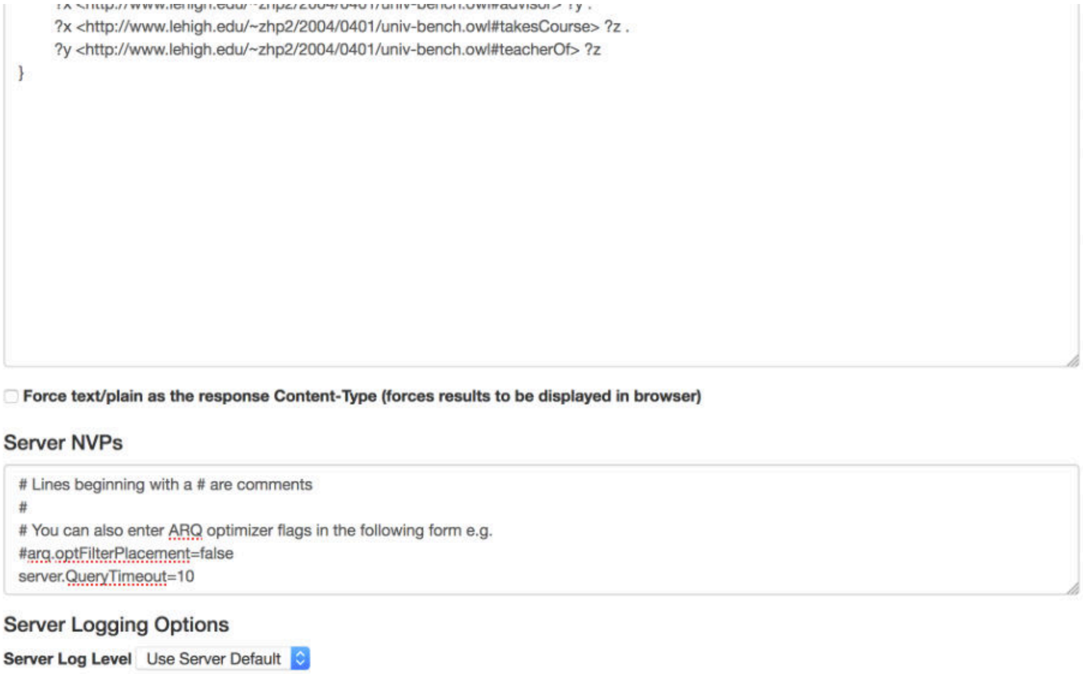
Terminate a CGE query using one of the following options

- Use the `server.QueryTimeout` NVP to cancel the query.

  This option can be used for cancelling a query by setting a timeout on a query by editing the NVPs to be sent with the query.

  1. Scroll down to the Server NVPs and set the value of `server.QueryTimeout` to the number of seconds the query should to be allowed to run before it times out.

Figure 17. Change the Query Timeout Value



```
    ?x <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#advisor> ?y .
    ?x <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse> ?z .
    ?y <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#teacherOf> ?z
}
```

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

Server NVPs

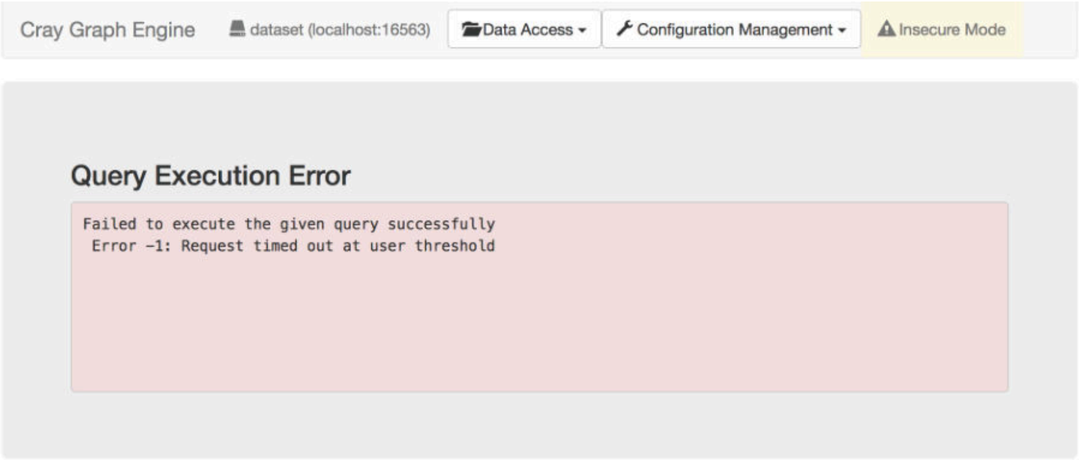
```
# Lines beginning with a # are comments
#
# You can also enter ARQ optimizer flags in the following form e.g.
#arq.optFilterPlacement=false
server.QueryTimeout=10
```

Server Logging Options

Server Log Level: Use Server Default

When a query runs more than that number of seconds specified via the `server.QueryTimeout` paramter , it will time-out, in which case, the UI will look similar to the following:

Figure 18. CGE Query Execution Error Pop Up



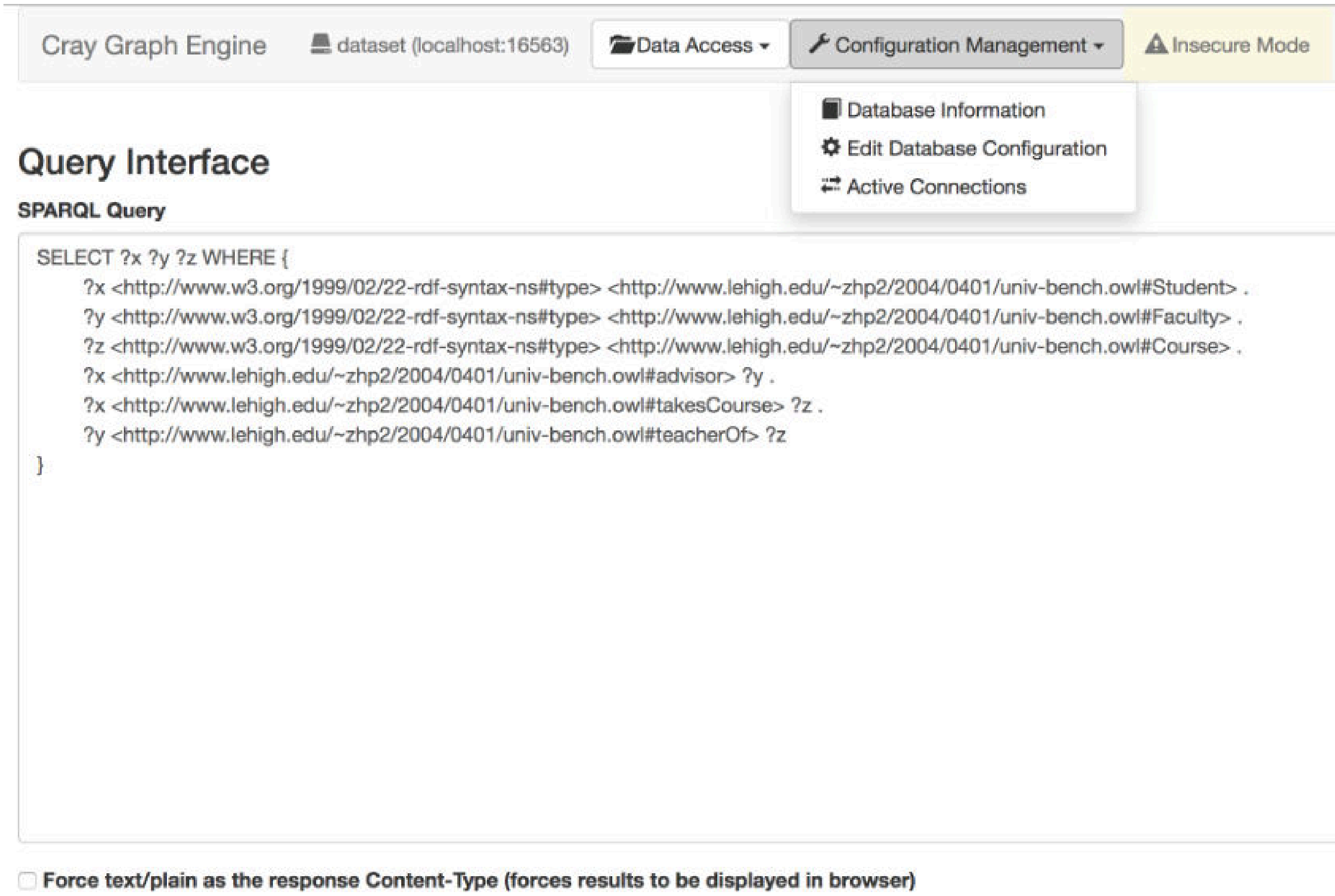
Query Execution Error

```
Failed to execute the given query successfully
 Error -1: Request timed out at user threshold
```

- Canel a query using the **Active DataBase Connections** screen.

  This option can be used for cancelling a query if a timeout was not specified via the `server.QueryTimeout` parameter.

  1. Select **Configuration Management** menu at the top of the window:



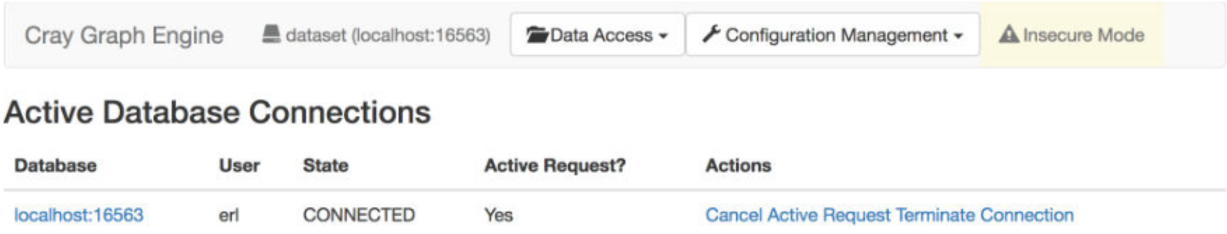
```
SELECT ?x ?y ?z WHERE {
      ?x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Student> .
      ?y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Faculty> .
      ?z <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#Course> .
      ?x <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#advisor> ?y .
      ?x <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#takesCourse> ?z .
      ?y <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#teacherOf> ?z
}
```

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

2. Select the **Active Connections** menu option.

The query under consideration will be displayed on this page as an active request.

Figure 19. CGE Active Database Connections Screen

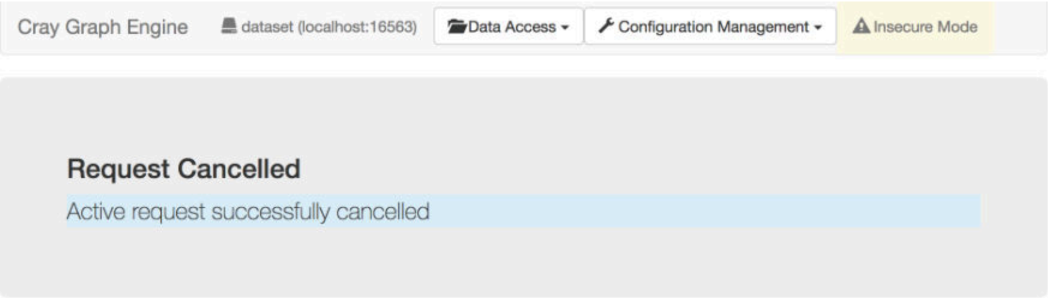


Active Database Connections

Database	User	State	Active Request?	Actions
localhost:16563	erl	CONNECTED	Yes	Cancel Active Request Terminate Connection

3. Select **Cancel Active Request** next to the currently running request.

Figure 20. CGE Query Cencelled Pop Up



Request Cancelled

Active request successfully cancelled

The query under consideration will be cancelled and the server will become available for other requests, usually within a few seconds.

6.2 Cancel a Request Running Under a CGE CLI Query

Queries submitted using the `cge-cli query` command can be terminated by using the `--nvp` option and specifying a timeout interval.

```
$ cge-cli query --db-port=16563 --nvp server.QueryTimeout 10 --quiet Query09.sparql
Error -1: Request timed out at user threshold
```

Queries can also be terminated at any time by simply killing the CGE CLI process by using `CTRL-C` or other signal:

```
$ cge-cli query --db-port=16563 --quiet mytests/lubm0/query/Query09.sparql; sleep 1; cge-cli echo --db-
port=16563
^C0 [main] WARN com.cray.cge.cli.CgeCli  - User data hiding is enabled, logs will obscure/omit user
data.  Set cge.server.RevealUserDataInLogs=1 \
in the in-scope cge.properties file to disable this behaviour.
1756 [main] INFO com.cray.cge.cli.commands.debug.EchoCommand  - Sending echo request...
1943 [main] INFO com.cray.cge.cli.commands.debug.EchoCommand  - Echoed data received and validated
successfully
```

7 CGE Security

CGE security starts at the entry point to the request handling in the CGE server and extends outward to the web UI and the CGE CLI commands. CGE Security is comprised of the following mechanisms:

- Server side user identification and authentication
- User permissions and access control
- User accountability
- Client side user identification and authentication

Server Side User Identification and Authentication

Users on the server side of CGE are identified by user names, which are character strings that name the user. User names within CGE are not necessarily tied to any specific user known to the Linux platform on which the CGE server is running, though there are scenarios in which it is practical to configure CGE users using their Linux login usernames. This freedom from the Linux platform permits a database owner to set up a CGE instance that is web accessible (more on this later) and has a user community completely defined by the database owner without respect to ability to log into the Linux platform on which the CGE Instance is running. This is similar to other web-based frameworks that permit the owner of the framework to set up the user community without needing to be able to create user logins on the host platform.

The CGE Server handles requests in the context of a client connection. Each connection establishes a context in which one or more sequential requests may be issued. While each connection may present a sequence of requests, these connections are not persistent in the sense that they represent an open ended logged in relationship with the client. The most common scenario is a connection that presents one or two requests and handles the responses, then disconnects.

Each connection is made without context preserved from any previous connection. Because of this, each time a client connects to submit requests, the client submits the user name (identity) of the user making the request. The CGE server uses the SSH public-key authentication protocol to verify that the client submitting the user name has the authorization to present that user name.

Normally, authentication strives to verify that the user presenting an identity actually *is* the user who owns that identity. In the case of the CGE server, the expectation is that this level of authentication has already been done on the client side. The CGE server needs to know that it is talking to a client that is authorized to present work on behalf of the specified user.

User Permissions and Access Control

The CGE server handles work as a sequence of requests. Each request has a particular type, such as

- Query
- Update
- Checkpoint
- Shutdown

Each request type has an associated permission that determines whether a client making that request is allowed to make that request or not. Permissions can be associated with individual users or groups of users by making permission assignments in an Access Control List (ACL) located in the directory where the CGE dataset is found.

When a request arrives, the username presented by the request is authenticated and then the permissions associated with that username are looked up. If the permission associated with the incoming request type is present in the user's permission set, the request is allowed to proceed. If not, the user is notified of the request failure and the request is not allowed to proceed.

This mechanism allows the owner of a CGE database to establish coarse grained protections against unauthorized actions by otherwise authorized user.

User Accountability

When a user submits a request, the CGE Server runs that request on behalf of that user. The owner of the CGE database may want to review the operations that have been executed by a given user. To this end, from the moment a request arrives to the moment that the request completes processing and reports its result (successful or not) the username of the client making the request is recorded with each log entry written by the CGE server into its operational log. Even if the user has the permission to turn off all logging for the duration of the request, CGE server records log entries at the beginning of the request indicating that the user has turned off logging. Those log entries are tagged with the requesting user's username.

Client Side Identification and Authentication

Client side identification and authentication is responsible for assuring that a user making a request actually is the purported user. There are two different kinds of client seen by CGE:

- logged in Linux users running CGE CLI commands and APIs as clients
- Web-based clients

The identification and authentication for these two different kinds of clients differs, so each are explored separately.

- **Logged In Linux users as clients** - A logged in Linux user has already been identified and authenticated by Linux, and the user's credentials have been established by Linux. If there were a simple trustworthy way to transmit those credentials directly to the CGE Server, this would be sufficient and the client would simply assert the user's logged in Linux identity with every request. Because requests are transmitted outside of a trusted context, however, the CGE Server authenticates the requested username using SSH public-key authentication as described above.

  Within the category of Logged In Linux users, a client may be either a normal client or a super client. The distinction is between clients that can only present a single username to the CGE server and clients that may present some larger set of usernames (constrained by the CGE server configuration) to the CGE server.

  - **Normal clients** - A normal logged in Linux user client presents the username of the Linux user with each request. The server side authentication of a logged in Linux user uses that user's public SSH key for public-key authentication of the user. Since Linux is a trusted repository for user identity, once the user has logged into Linux the user's identity can be trusted (by the client) at all times. The degree to which the client is trusted by the CGE Server hinges on the ability of the SSH protocol to match the requested username with a working public key.
  - **Super clients** - A super client is a logged in Linux user whose private SSH key matches with more than one public-key/username pair in the CGE Server configuration. Generally, the owner of the CGE Database will be a super client, allowing him or her to run the Web UI and enable user authentication, but any user can be set up by the database owner as a super client. In the case of a super client, an arbitrary

username is presented with each request (generally corresponding to a user who has been authenticated using some higher level mechanism). If the username matches a public key that works with the super client user's private SSH key, the supplied username will be used by the CGE server. If not, the request will fail to authenticate at the CGE server and will not proceed.

It is worth noting that the use of the same public key for multiple users while keeping the associated private key private to the owner of that key does not constitute SSH key sharing, since there is only one user (the super client user) who owns the key pair. In the case of key sharing, all users sharing the key have access to the same key pair. In this case, only the super client has access to the private key and the public key is used to allow the super client to authenticate as 'authorized to present' the specified username.

- **Web UI clients** - The Web UI, CGE CLI front end is also capable of authenticating clients. It supports authentication using either an enterprise LDAP server or a user private authentication mode. The Web-UI also permits encryption of web transactions using SSL, to protect secrets (both authentication secrets and data secrets) in transit. When a user logs into the Web UI, the Web UI presents the logged in user name instead of the username of the Linux user who started the Web UI. For this to work, the user who starts the Web-UI needs to be the super client who has the correct private SSH key for all of the Web based users authorized to use the CGE Instance.

  Notice that Web UI clients are separated from logged in Linux clients by the keys used to log them in. A Linux user who has Web UI username/key pair on the CGE Server but no Linux username/key pair cannot use the Linux command line CGE CLI command. By the same token, if the user has no Web UI username/key pair, that user cannot use CGE through the Web UI. This allows the CGE Database owner to control both the form of access (via permissions) and the mode of access (command-line or web or both).

7.1 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.

7.1.1 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

Create the key in the `.cge` directory using `ssh-keygen(1)` instead of creating the key in the `.ssh` directory:

```
$ 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
```

> **NOTE:** While this example shows creation of keys under `$HOME/.cge`, it can be used to place keys in any directory. If `$HOME/.cge` is not a convenient place to put the keys, follow the above procedure to generate the keys in some other (suitably protected) directory, then use the `--configDir` option to `cge-launch` or the `$CGE_CONFIG_DIR_NAME` environment variable to point to that directory. If it is required to use CGE specific keys that are stored on HDFS, create them in a temporary directory using this procedure, then copy them onto HDFS in the location of your choice (appropriately protecting them). Then use an HDFS URL as the value of `$CGE_CONFIG_DIR_NAME` or the argument to the `--configDir` option to the `cge-launch` command to select that directory instead of `$HOME/.cge` as the key directory.

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

7.2 Setup CGE Security

Setting up security for a given instance of CGE involves the following steps:

- Configure authorized logged in Linux users (including the database owner) in an appropriate `authorized_keys` file
- Configure any super client authorized users using the public SSH key of the Super Client and the usernames of the various users in the appropriate `authorized_keys` file
- Configure user permissions in the database ACL file
- Optionally create an SSL configuration for `cge-cli fe`
- Create an authentication configuration (private or LDAP, simple or forms based) for `cge-cli fe`
- Start the CGE Web UI using `cge-cli fe` with appropriate security options

7.2.1 Configure Server Side Identification and Authentication

Configuring server side identification and authentication includes setting up both authorized Linux logged In (i.e. command-line) users, and setting up any super client authorized users that are needed for Web UI access or other purposes. The database owner needs to make decisions about the following:

- Is it required to allow command line clients to access the dataset?
- Is it required to grant Web-UI clients access to the dataset?

- At what scope (single dataset or all the datasets) is it required to authorize each given user, both the Web-UI identity and the Linux identity?

The `authorized_keys` File

The CGE Server searches the file named *authorized_keys* in each of the following directories for a username that matches the username presented with a given request:

- the database data directory
- the CGE configuration directory (either by default: `$HOME/.cge`, or the value of `$CGE_CONFIG_DIR_NAME` which can be set using the `--configDir=`*`URL`* option to `cge-launch`)
- `$HOME/.ssh`

Each username match is tested in turn until the public key associated with that match works for public-key authentication. Once a match is found, the user has successfully authenticated and becomes an authorized user for the duration of that request. Subsequent permission checks determine what that user is authorized to do.

One important decision the database owner needs to make is where to put a given user authorization. The choice of the `authorized_keys` file to store a given username/key combination depends on the breadth of authorization the owner of one or more databases wants to grant to the user. This breadth has three scopes:

- authorization to use only one database
- authorization to use all databases configured from the same configuration directory (typically all owned databases)
- authorization to use all owned databases and, likely, to log into the Linux host using the self identity

By placing a user's authorization in the `authorized_keys` file in the database data directory, the user is granted the most limited scope of authorization. This is appropriate for users that need to be granted access to that specific database, or if the database owner owns multiple databases with multiple potentially overlapping lists of authorized users and wants local control over each user authorization. By placing the user's authorization in the CGE Configuration directory, the user is granted intermediate scope of authorization. This is appropriate if the database owner owns multiple databases with a core set of users who are authorized on all of the databases. Placing a user in the `$HOME/.ssh/authorized_keys` file can potentially permit that user to log in as the database owner, which is a serious security threat. Never put any username/key combination that is not specifically your own SSH key (for login) in the `$HOME/.ssh/authorized_keys` file. This file is included in the search solely to make using CGE as the owner of the database simpler.

Authorized Keys for Command Line/API Use of the CGE Database

Users who are authorized to log into the server where the CGE database resides and use the `cge-cli` command or one of the CGE APIs to interact with your CGE Database need to have the public SSH key corresponding to their private SSH key stored in one of the `authorized_keys` files. Users may communicate their public SSH keys to you using whatever means (E-mail, publicly readable files, etc.) is mutually convenient. The user's public key can be generated using the `ssh-keygen` command and usually resides in the user's `$HOME/.ssh` directory in one of the following files:

- `id_rsa.pub` (RSA based public key)
- `id_dsa.pub` (DSA based public key)

When adding a command line/API key to the *authorized_keys* file, make sure that the key is a single line comprised of three parts (separated by spaces):

- the signing scheme used by the key (RSA or DSA)

- the key itself
- the username@host identifying the user

and that the username part of the username@host part matches the logged in user name of the requesting user. The host part is ignored, so it can be anything. Here is an example entry. Note that the content, which is a single line, is split up into multiple lines due to lack of space:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDOVyLTKwz/RAngMegeTST2Ow0JMwFea9qQC6R7en7A+BcsIaNt2m
+9Vh/AocMfaruwpyHr26\
epsdpC8Thw4+9NIUfoUoJyKC6TMZcntF7e3RiY1yZt6uvKUIgs75zS4fqZMAtHEiuvgLHkZwypKF1vsscu
sSYCMkNxXUa0E38UcPVmH\
+zEGWpc9yyObl+7Ae4PuKIjw6gpOtX8W8Wz/
Eb5UAwf56pCR045izZBwRe7y9anHe3+XtluFU9zU1I80aeRHg64KmMS3jCNhGIFOwmW\
O8iYmxHXyCheifxdYpCgI+jN+jQ6CqbFe4OrbkbuP/elAmFYl5BHMWi7LmYVWEYP user@nid00030
```

This will authorize the user user with the corresponding private SSH key to use your database.

Authorized Keys for Web-UI Users

The Web-UI uses the Super Client key of the user running the **cge-cli fe** command to submit requests on behalf of logged-in Web-UI clients. Normally, the user running the **cge-cli fe** command is the owner of the database, so examples of adding users are shown using your public SSH key as the authorizing key for Web-UI users in the *authorized_keys* file.

Assuming you are the user who will be running the **cge-ci fe** command for your database, the following command adds the user **david** as an authorized Web-UI user of your CGE Database:

```
$ $ sed -e "s/ $USER@/ david@/"< $HOME/.ssh/id_rsa.pub >> \
    authorized_keys
```

This replaces your username from your id_rsa.pub public key file (a similar command will work with an id_dsa.pub file as well) with **david** creating a user named **david** that you are authorized to authenticate for your CGE Database instance. Do this for all the Web-UI users you want to authorize. You will also need to make sure they are able to log into your Web-UI.

7.2.2 Configure the ACL File User Permissions

By default, in the absence of an ACL file, users of the CGE database file will fall into one of two categories:

- the instance owner (a user who's username matches that of the Linux username of the user who started the CGE Server)
- everyone else

As the instance owner, you have all permissions when interacting with the CGE Server. All other authorized users are permitted only to query the database.

This section explains how to set up an ACL file that allows more precise control of access to the database on a per-user basis.

CGE Permissions

CGE uses a hierarchical set of permissions to control the types of requests an authorized user is permitted to make. The following lists the permissions and the requests or actions they control:

- `data.query` - permission to query (read only) the data set
- `data.update` - permission to update (write only) the data set
- `data.checkpoint` - permission to checkpoint (save to storage) the data set
- `request.nvp` - permission to set one or more configuration NVP settings to be effective for the duration of an individual request, if not present, specifying NVP settings causes the request to fail
- `request.log` - permission to modify logging behavior for the duration of an individual request, if not present, logging is unchanged but the request proceeds normally
- `server.config.nvp.get` - permission to read the NVP configuration in effect in the running server
- `server.config.nvp.set` - permission to alter the NVP configuration in effect in the running server for all subsequent requests
- `server.config.log.get` - permission to read the logging configuration in effect in the running server
- `server.config.log.set` - permission to alter the logging configuration in effect in the running server for all subsequent requests
- `server.config.output.get` - permission to read the name of the output directory used by the server to store result files
- `server.config.output.set` - permission to change the output directory used by the server to store result files for all subsequent requests
- `server.shutdown` - permission to shut down the running server

Permissions may be named individually or may be named using a wildcard character ('*') at any level of the hierarchy. A wildcard character all by itself signifies all permissions. Here are some examples of permission sets and their equivalent wild card definitions:

- All Permissions : *****
- `data.query,data.update,data.checkpoint` : `data.*`
- `server.config.nvp.get,server.config.log.get,server.config.output.get` : `server.*.get` or `server.config.*.get`
- `server.config.nvp.get,server.config.nvp.set` : `server.*.nvp.*` or `*.nvp.*, or server.config.nvp.*`

Notice that various more or less specific forms of wildcarding produce the same result with the current set of permissions. In future releases, new permissions might be added that might match one of the less specific wildcard specifications and grant unexpected rights to a given user. It is generally best to use the most specific wildcard form possible to achieve the desired set of permissions so that you do not experience permission creep from release to release. It is also a good idea to review you ACLs with respect to the permissions available in a given release to ensure that no new permissions are being unexpectedly granted.

The ACL File

The ACL file is a file named *user_perms.cfg* in the data directory of your database. This file, if present, contains the permission assignments for authorized users using your database. For your convenience, this file allows you to group permissions and users using `roles`, each of which is a named set of permissions containing the permissions needed to carry out a specific related set of database responsibilities, and **groups**, each of which is a named set of users to be assigned a common set of permissions or roles. The ACL also permits you to specify permissions or roles for individual users by name, and to specify a default set of permissions using the **default user name** (*****). Here is a sample ACL that illustrates all of these elements:

```
[roles]
# An administrator has all permissions
admin = *

# An auditor has the ability to adjust the logging
# behavior of the running CGE Server
auditor = server.config.log.*

# A consumer of data is allowed to query the CGE Database
# and provide per-request NVPs because some NVPs impact the
# efficiency / practicality of certain queries.
consumer = data.query,request.nvp

# A producer of data is allowed to query, update, and
# checkpoint the CGE Database, and is allowed to set
# per-request NVPs because some NVPs impact the efficiency
# or practicality of certain queries, and some options on
# checkpointing are controlled by NVPs.
producer = data.*,request.nvp

[groups]
admins = joe,mary,abdul
auditors = phyllis,jodi,allan
producers = anne,grace,william

# A group of users to whom no access is permitted.  This is
# a useful way of temporarily disabling a user while keeping
# that user's authorized keys active.  The group is defined
# here with its member list, but is never assigned any roles
# or permissions. This prevent's these users from being
# treated as default users (allowing default users to have
# more permissions) while ensuring they have no access.
denied_users = wilbur,ginger,ava

[permissions]
group:admins = role:admin
group:auditors = role:auditor
group:producers = role:producer
# The user 'david' is an auditor who also needs to be able to
# see what is in the database, so he needs both the auditor and
# consumer role.
david = role:auditor,role:consumer
# The user 'fred' needs to be able to query, but we don't trust
# him with changing per-request NVPs, so he can't do that.
fred = role:auditor,data.query

# Everyone else who is not specifically mentioned either by name
# or by group is allowed to be a 'consumer'
* = role:consumer
```

There are a few things to notice about the above sample ACL. First of all, it is divided into sections of three types:

- The `roles` section contains role definitions
- the `groups` section contains group definitions

- the `permissions` section contains permission assignments to both groups (where the group name is qualified by the `group:` prefix), and users.

There may be any number of sections of a given type. The aggregate effect of multiple sections of the same type is the same as having one large section of that type containing all of the content of the smaller sections.

The second thing to notice is that comments are permitted in an ACL file. Comments take the form of a '#' character followed by any arbitrary text up to a newline. The comment ends at the newline.

The third thing is not obvious from the example, but the sections, definitions and assignments do not need to be presented in any particular order. As long as the definitions and assignments take place within the appropriate sections and convey an unambiguous intent, the CGE Server will figure out any necessary ordering.

There are some rules about what constitutes unambiguous intent:

- A role or group may have at most one definition in the ACL
- A group or user may have at most one permission assignment in the ACL
- A user may belong to at most one group
- A user may not both belong to a group and have a permission assignment
- The list of permissions and roles in a permission assignment may contain any arbitrary list of permissions and roles, even repeated permissions or repeated roles
- The default (*) user is a default user, not a wildcard user, so assigning permissions or roles to it, at most once, does not violate any of the above rules regarding explicitly named users

If an ACL file is changed while the CGE Server is running, its contents will take effect upon receipt of the next request. This permits on-the-fly changes to the ACL, but it also opens up the possibility of creating a malformed interim ACL while editing or in the process of copying a new ACL into position. In order to replace an ACL safely, it is a good practice to make a copy of the ACL that needs to be edited, then edit the copy and verify it using the `cge-test-permissions` command, before *moving* it into place using the `mv(1)` Linux command. The advantage of using the `mv(1)` command instead of the `cp(1)` command is that the rename of the file performed by `mv(1)` is atomic, so no request can come in while the file is being copied. The risk of a race condition here is tiny, but it could produce surprising effects that cannot be reproduced.

For more information, see the `cge-test-permissions(1)` and `CGE-PERMISSIONS(5)` man pages.

ACL File Verification

The `cge-test-permissions` command allows you to verify the correctness of an ACL without needing to read through the CGE Server log for errors. While an ACL file may reside on any file system accessible by the CGE Server for its ultimate use (e.g. HDFS) , the `cge-test-permissions` command only has access to files on POSIX compliant (i.e. Linux native) file systems. Since you are most likely to edit your ACL files on a native file system and then copy them to, for example, an HDFS file system, this should not be too much of an inconvenience, but it is important to note that specifying a URL for an ACL filename to `cge-test-permissions` will result in an error.

Here are a few examples of common uses of `cge-test-permissions` using the example ACL file shown above:

```
# Check that the ACL file is correctly formed and unambiguous,
# expecting a silent exit (the exit value will be 0) on success
$ cge-test-permissions perms_example.cfg

# Check the definition of the 'auditor' role
```

```
$ cge-test-permissions -r auditor perms_example.cfg
Role 'auditor':
        Permissions: server.config.log.get, server.config.log.set

# Check the definition and permissions assigned to the 'auditors' group
$ cge-test-permissions -g auditors perms_example.cfg
Group 'auditors':
        Roles:
            auditor [server.config.log.get, server.config.log.set]
        Assigned Permissions: <none>
        Effective Permissions: server.config.log.get, server.config.log.set
        Members: phyllis, jodi, allan

# Check the definition of the user 'jodi'
$ cge-test-permissions -u jodi perms_example.cfg
User 'jodi':
        Member of group auditors [server.config.log.get, server.config.log.set]
        Roles: <none>
        Assigned Permissions: <none>
        Effective Permissions: server.config.log.get, server.config.log.set

# Check the definition of the user 'fred'
$ cge-test-permissions -u fred perms_example.cfg
User 'fred':
        Member of no group
        Roles:
            auditor [server.config.log.get, server.config.log.set]
        Assigned Permissions: data.query
        Effective Permissions: data.query, server.config.log.get,
server.config.log.set
# Check the definition of the denied user 'ava'
$ cge-test-permissions -u ava example_acl
User 'ava':
        Member of group denied_users []
        Roles: <none>
        Assigned Permissions: <none>
        Effective Permissions: <none>
```

If any of the above commands were run using an ACL with errors or ambiguity in it, the command would have reported errors as it found them, allowing you to correct the errors and re-run the command.

The first example is simple. If no options are given, the command simply verifies that the specified file is acceptable and exits silently if it is okay.

The second example displays the contents of the single role named `auditor`. Here the name of the role and the permissions that make up that role are displayed.

The third example displays both the definition and the permissions of the group `auditors`. In this case, the name of the group and the users making up that group are displayed. In addition to that, though, the group also may be assigned some set of roles, and may be assigned some set of explicit permissions. Any roles assigned to this group are displayed, and two different kinds of permissions are displayed. The first set of permissions is the `Assigned Permissions` these are the permissions that were explicitly assigned to the group by name. The second set of permissions is the `Effective Permissions`. These are the permissions that result from combining the permissions derived from roles with any permissions explicitly assigned by name. They are the permissions that will actually be used to make an access decision when a member of this group issues a request to the CGE Server.

The fourth example displays information about the user `jodi`. A user may belong to a group, or be assigned permissions and / or roles explicitly, so all of this is displayed. Here we see that `jodi` is a member of the group `auditors` which contributes a set of permissions, but has no explicitly assigned permissions or roles. The `Effective Permissions` here are the permissions derived from group membership, role assignment and explicit permission assignment. In the case of `jodi` the effective permissions are derived from the group `auditors` so they are the same as that group.

The fifth example displays information about `fred` who has an explicit role assignment and an explicit permission assignment but is not a member of any group. Here we see the role `auditor` contributes a set of permissions, and there is one permission explicitly assigned to `fred`. The `Effective Permissions` in `fred`'s case are the combination of the role permissions and the explicit permissions (no permissions are contributed by a group).

The last example displays information about `ava` who has had all permission explicitly denied to her by placing her in a group of users who are denied permissions. Notice that she is a member of the group `denied_users` which has no permissions assigned to it. She has no explicit permissions and no effective permissions.

It is also possible to dump out the complete state (all roles, groups and users) defined by an ACL file using the`cge-test-permissions -a` command. This produces a lot of output, so it is not shown here, but the output is formatted the way it is shown above.

Once you are satisfied with your ACL file, place it in your data directory and it will take immediate effect.

7.2.3 Configure Web UI Identification, Authentication and Encryption

After setting up the web UI users and their permissions, the web UI needs to be configured to identify and authenticate users who want to use the database through the web UI. The `cge-cli fe` command searches the working directory from which it is launched followed by the CGE configuration directory (either by default: `$HOME/.cge`, or the value of `$CGE_CONFIG_DIR_NAME` which can be set using the `--configDir=`*`URL`* option to `cge-launch`) to find its configuration files. Any configuration that is put in the CGE configuration directory will be shared by any web UI that is launched using that directory. This can be convenient when running multiple web UI instances using the same configuration.

There are three major elements of this configuration:

- The identification and authentication mechanism to be used (private or LDAP) and the form in which the credentials are presented to the CGE web UI (forms or HTTP basic).
- The list of users and passwords to use (or, in the case of LDAP, the particulars of the LDAP server interaction)
- SSL Encryption to protect the content of communications (including credentials) and to assure the web UI user of the authenticity of the web UI service.

Choose and Configure an Identification and Authentication Mechanism

The web UI uses the Apache Shiro Security Framework to implement Identification and authentication of users. This framework permits the user to configure one or more security *realms* as the basis for Authentication. An example of a realm is LDAP authentication, in which an enterprise or cluster based LDAP directory is used for authentication decisions. Another example is a simple private text based user / password list stored in the configuration (.ini) file. Which of these you choose depends on how you want to define your user base for your instance of CGE.

In addition to this, there are two different mechanisms for collecting the user's identity and authentication credentials: HTTP Basic and Forms based. In the HTTP basic approach, HTTP issues an authentication challenge to the browser or application attempting to access your web UI and the browser or application prompts the user for an identity and a password. From that, the browser generates and remembers a set of authentication credentials and attaches them to every subsequent request. This is useful for programmatic access to the web UI, but can be a bit cumbersome for user interactive use. In the Forms based approach, the user is presented with a login page on first contact with the web UI. The user fills out a username and password, and the web UI establishes a session with the user. This is very convenient for interactive use of the web UI but awkward for programmatic use, where the program will have trouble interacting with the login page. Which of these you choose depends on the mix of user interactive and programmatic access you expect your web UI to support.

CGE offers a tool, `cge-cli generate shiro`, that allows you to generate template configuration files for HTTP Basic with private authentication data, Forms based with private authentication data, and Forms based with LDAP authentication. To use HTTP Basic with LDAP you need to make a minor change to the Forms based with LDAP configuration.

More complex and expressive Shiro configurations are also supported by CGE, but `cge-cli generate shiro` does not offer tools to generate templates for them. Templates and advice may be found in the open-source Shiro community. For more information, see *http://shiro.apache.org/documentation.html* .

Choose the Mechanism

The first choice that needs to be made is the kind of authentication the CGE Instance web UI needs. If the user is setting up an independent instance of CGE where the user wants to fully control the security environment of the instance, or the user does not have access to an LDAP server that fully expresses the range of users the user will be interacting with, then the private approach to storing authentication data makes sense for the CGE Instance. An example of this might be some kind of moderated public access to a CGE Database, where the user does not want other users to be configured as part of the user's LDAP directory. If the user is setting up an enterprise wide CGE server, where the authentication data for all users is already stored in an enterprise LDAP server, and it is required to allow those users selective access to the CGE Instance, the LDAP approach makes the most sense. The ability to log into the web UI does not necessarily impart the ability to interact with the database. The user must also be authorized as a web UI client (i.e. have the web UI Super Client public key associated with his or her username in an `authorized_keys` file).

The next decision is whether to use the HTTP or Forms based login mechanism. If it is expected to have a mix of user interactive and programmatic use of the web UI, then HTTP Basic makes the most sense, even though it is a bit more cumbersome for interactive users. If only user interactive use is anticipated, then the Forms based approach makes the most sense.

7.2.4 Configure LDAP for CGE

To set up an LDAP based Apache® Shiro template configuration file, issue the following command on the login node of the system where it is intended to run CGE:

```
$ cge-cli generate shiro ldap > $HOME/.cge/shiro.ini
```

This command will create a template configuration that you can edit to work with your specific site LDAP server. If running CGE on a Urika-GX system, Cray recommends to have a centrally configured LDAP server for the Urika-GX cluster running on the login node in order to use that LDAP server as a forwarding agent to the site's enterprise LDAP. To use this approach for configuring LDAP for CGE, change the following line in the configuration to include the name of the login node instead of `host-login1`:

```
ldapRealm.contextFactory.url = ldap://host-login1:389
```

For example:

```
ldapRealm.contextFactory.url = ldap://machine-login1:389
```

In the preceding example, *`machine`* is used as an example for the name of the machine. This tells Apache Shiro where to look for the Urika-GX LDAP server, which resides on your Urika-GX login1 node as Urika-GX is shipped.

The configuration that results here will be Forms-based. To use an HTTP basic configuration with LDAP, change the following line:

```
/** = authc
```

to:

```
/** = authcBasic
```

This will make the default requirement for accessing Web-UI pages be HTTP basic authentication (`authcBasic`) instead of Forms authentication (`authc`).

CGE web UI can be directly integrated with the site's LDAP server, in which case, the configuration will need to match what the site's LDAP expects. To enable this, edit the part of the template that looks like:

```
# Define a LDAP realm
ldapRealm = org.apache.shiro.realm.ldap.JndiLdapRealm

# Configure the template for User lookups
# You will need to ask a system administrator what the format should be here
# The following is the default on Urika-GX systems as shipped but your system
# may be differently configured
ldapRealm.userDnTemplate = uid={0},ou=People,ou=external,dc=local

# Configure to point to LDAP server of choice
# The LDAP server resides on the login1 node on Urika-GX systems as shipped
# 389 is the normal default port for LDAP servers
ldapRealm.contextFactory.url = ldap://host-login1:389

# Only uncomment and change this if your server needs a specific auth mechanism.
# By default the client should negotiate this automatically with the server
#ldapRealm.contextFactory.authenticationMechanism = DIGEST-MD5

# If your LDAP server needs credentials to access it set them here
# In most cases this should be unnecessary
#ldapRealm.contextFactory.systemUsername = ldap-admin
#ldapRealm.contextFactory.systemPassword = ldap-admin-password
```

7.2.5 Configure Private Authentication for CGE

About this task

Use this procedure to set up private authentication for the CGE instance web UI.

Procedure

1. Execute one of the following commands on the login node of the system where CGE is intended to run.

- `$ cge-cli generate shiro simple > $HOME/.cge/shiro.ini`
- `$ cge-cli generate shiro forms > $HOME/.cge/shiro.ini`

  The first command will produce an HTTP Basic configuration template, the second command will produce a forms-based configuration template.

2. Add users.

a. Look for the users section of the configuration template.

```
[users]
# Define two users
admin = admin
user = password
```

b. Edit users as needed.

For example, to have three users: `phyllis`, `jodi` and `allan`, set up the users as follows:

```
[users]
# Define two users
phyllis = PasswordForPhyllis
jodi = PasswordForJodi
allan = PasswordForAllan
```

These examples show the passwords stored as clear-text. Refer to *http://shiro.apache.org/documentation.html* for examples related to using one-way encryption to make passwords less accessible .

7.2.6 Configure SSL for CGE

SSL provides three types of protection for data and users. The first protection it affords is assurance that the user is interacting with a web UI that is, in fact, the web UI for your CGE instance. By providing an SSL certificate that is correctly signed, your web UI tells users (and their browsers) that they are talking to the right web UI. To the user, that means that it is safe to present a username and password to the web UI, that the user can safely present sensitive information to the web UI without concern that an impostor web UI will steal it, and that any data coming from the web UI is trustworthy, since it comes from a verified web UI. This means that the user can trust the data for decision making and trust the database with new data. The second protection SSL affords is encryption of authentication secrets so that the user can present a username and password without fear of these secrets being intercepted in-flight to the web UI. The third protection SSL affords is encryption of query and update data so that the user can query and update the database without fear of sensitive query results or sensitive update data being either intercepted or modified in flight to and from the web UI.

There are two kinds of SSL certificates that may be used to provide SSL protection of the web UI:

- Verified - A verified SSL certificate is purchased from a third party Certificate Authority (CA). The CA provides a secure verification service. Certificates from that authority can be verified securely by any web browser or SSL enabled application with no user intervention.
- Self-signed - A self-signed certificate is one that the owner of the web UI can generate for themselves, but which has no third-party verification. Users are prompted by their browsers to accept or reject self-signed certificates, and are usually advised not to accept them. In some cases, where users know for sure what your certificate looks like and that you are trustworthy, they might be willing to accept a self-signed certificate. In general, self-signed certificates are used for prototyping and debugging of web UI deployments. When it comes time to go live with data, it is a good idea to obtain a verified certificate and replace the self-signed

certificate with it. CGE provides the `cge-cli generate keystore` command to help with creation and importation of SSL certificates.

- **Using a Verified SSL Certificate** - Obtaining a verified SSL certificate is outside the scope of this discussion, but once you have an SSL Certificate downloaded to your site and want to install it, installation is simple. The following command will import the certificate into your keystore for you to use:

  ```
  $ cge-cli generate keystore --import your.cer --keystore ~/.cge/keystore
  ```

  This will produce a file named `keystore` in the `.cge` directory in the home directory. This is the default place that CGE looks for CGE configuration files. The keys in the `keystore` file will be found by CGE by default by looking in this file. If a different directory is used (or, for example, a directory on HDFS) for CGE's configuration, it is possible to choose the path or URL of that directory as the argument to the `--keystore` option. The SSL certificate will be imported from the file *`user`*`.cer` which is the verified certificate downloaded from the certificate authority.

  ⚠ **CAUTION:** In order for the imported certificate to be usable it must contain the private key as well as the Digital signature from the certificate authority. Without the private key a certificate cannot be used for SSL

- **Using a Self-Signed SSL Certificate** - To use a self-signed certificate, execute the following command:

  ```
  $ cge-cli generate keystore --self-signed --keystore ~/.cge/keystore
  ```

  The system will be prompt for a bunch of information about the self-signed certificate and then it will be created in the `.cge` directory in the home directory. This is the default place that CGE looks for CGE configuration files. The keys in the `keystore` file will be found by CGE by default by looking in this file. If using a different directory (or, for example, a directory on HDFS) for CGE's configuration, it is possible to choose the path or URL of that directory as the argument to the `--keystore` option

- **Giving Your Web-UI Access to Your SSL Keystore** - In addition to file protections, both the SSL keystore and certificates can be password protected. In this case, `cge-cli fe` needs to know these passwords to access the certificate. These passwords need to be stored in the CGE `properties` file (by default `$HOME/.cge/cge.properties`) as follows:

  ```
  cge.cli.server.ssl.password = MyKeyStorePassword
  cge.cli.server.ssl.key-password = MyCertificatePassword
  ```

  By default these passwords are stored in clear text. If you want them stored in an obfuscated (one-way hashed) form, you can use the following command to set up these passwords:

  ```
  $ cge-cli generate properties --ssl-passwords
  ```

  The system will prompt for these two passwords, obfuscate them, and add them to the `cge.properties` file.

- **Securing Your SSL Certificate** - The SSL certificate contains sensitive information and should be properly secured. With it, it is possible for an impostor to impersonate the SSL protected web-site. While the information in the SSL keystore is somewhat obfuscated, it is best not to treat it as secured simply by those means. Using Linux file permissions you can further secure the keystore to help prevent unauthorized use. If a user needs to run the Web-UI (i.e. invoking the `cge-cli fe` command) the user can simply make the file mode readable only by themself. For example:

  ```
  $ ls -l keystore
  -rw-r--r-- 1 erl criemp 2222 Sep 26 10:56 keystore
  $ chmod 600 keystore
  $ ls -l keystore
  -rw------- 1 erl criemp 2222 Sep 26 10:56 keystore
  ```

Take similar steps to protect the `cge.properties` file and any verified certificate files, since these contain similarly sensitive data.

7.2.7 Launch a Secured Web UI

Prerequisites

Set up the CGE authentication and SSL encryption

About this task

After setting up the authentication and SSL encryption in the desired way, launch the Web UI using the configured security features.

Procedure

Launch the CGE web UI

- If the filenames used in the security section examples of this publication are used, and the CGE database instance is running on the default port, the following command will start a secure version of the Web UI with both authentication and SSL encryption enabled:

```
$ cge-cli fe --security=/data/directory/shiro.ini --ssl
```

- If not using SSL, then the following command will enable authentication without SSL.

⚠ **CAUTION:** There is a chance of credential leakage when not using SSL, so this is not really a secure way to run a web UI.

```
$ cge-cli fe --security=/data/directory/shiro.ini
```

7.3 Endpoint Security

The CGE server you provides two mechanisms for endpoint security:

1. SSL encryption
2. User authentication

Both of these features are off by default as they require additional user provided configuration.

SSL Encryption

When enabled, SSL provides encryption of communications between the client and the SPARQL server. Note that Communications between the SPARQL server and the database server are always encrypted regardless of whether this is enabled. By enabling this feature you gain complete end to end encryption from the client all the way to the database server.

SSL Certificates and the Key Store

In order to enable this feature you will need to provide a suitable SSL certificate. Obtaining an SSL certificate is covered elsewhere in the documentation and you should refer to that for more detail.

To use a certificate, import it into a Java key store, which can be done using the Java `keytool` utility:

```
$ keytool -import-v -trustcacerts -alias cge -file server.cer -keystore ~/.cge/keystore
```

In this example we import a certificate from the `server.cer` file to the key store located at `~/.cge/keystore`.

Note that each certificate must have a unique alias within the key store. Key stores are protected by a password for which the user will be prompted. If the key store does not yet exist, the user will be prompted for a new password and a new file will be created.

In order for the server to pick up the correct certificate, the key store file should ideally contain only the certificate to be used for SSL. If it contains multiple certificates, the SSL may fail to function.

The `cge-generate keystore` command provides a simple wrapper around some common `keytool` commands. For example, the above could also be execute as:

```
$ cge-cli generate keystore --alias cge --importserver.cer --keystore ~/.cge/
keystore
```

Locating the key store

For the server to access the key store, it must be provided with the location of the key store and the password to access it. The location can be provided explicitly using the `--keystore` option. if this is not specified, then it tries to locate a key store as follows:

1. If the located configuration file contains a `cge.cli.server.ssl.keystore` property, use the file specified by that.
2. Otherwise search for a file named `keystore` in any of the specified configuration directories. As with other configurations files only the first one found will be used.

   If the key store and/or the certificate itself require passwords then these must be provided in the configuration file using the following properties:

   Table 15. Key store Properties

Property	Description
`cge.cli.server.ssl.password`	Password for the key store
`cge.cli.server.ssl.key-password`	Password for the certificate

   To avoid storing the password in plain text it may be stored in obfuscated form as supported by Jetty.

Enabling SSL

Assuming the appropriate certificate is in place in the key store, and the properties file is configured with any necessary passwords then, enable SSL using the `--ssl` option, as shown in the following example:

```
$ cge-cli fe --ssl
```

This will start the server configured for SSL Communications i.e. It will only respond to `https://` URLs. Communication with the server will not be possible without an appropriate certificate.

Enabling Lax SSL

The default configurations for SSL only permits strong cipher suites and cryptographic protocols to be used. Some older tools may encounter difficulties when trying to communicate with the server if they do not support appropriate cipher suites and/or cryptographic protocols. In this case you may want to enable Lax SSL mode. For example:

```
$ cge-cli fe --ssl --ssl-lax
```

⚠ **CAUTION:** In this mode, the server will permit the use of cipher suites and cryptographic protocols that have known flaws, are considered weak and/or may be susceptible to widely published and easily reproducible attacks. Therefore, we strongly recommend that you only use this mode when absolutely necessary.

7.4 CGE User Authentication

User authentication allows you to configure the server such that users accessing it must first authenticate themselves. This means that the server knows the identity of the user and can provide this information to the database server meaning that only users authorised to access the database can perform actions against it. This provides for a strong audit trail that logs user activity on a database.

When not enabled the server runs in anonymous access mode. This allows anybody to access the server and all actions are carried out using the identity of the process owner.

Apache Shiro configuration

The server relies upon Apache Shiro to provide the authentication layer, this allows for a wide range of configurations that can be tailored to your requirements. Note that authentication does not imply authorisation, it is perfectly possible to create A configuration where a user can authenticate themselves but does not have the authorisation to actually perform actions against the database.

In order to enable authentication you must provide a valid Shiro configuration file, which is beyond the scope of this publication.

For more information, see *Generate a Shiro Configuration Template Using the generate shiro Command* on page 49 and visit *http://shiro.apache.org/configuration.html* and *http://shiro.apache.org/web.html*. As many users may not be familiar with this framework the command line interface includes a helper command that will generate templates for the most common configurations.

Enabling user authentication

Once the appropriate Shiro configuration has been put in place, the user can start the server with authentication enabled, as shown in the following example:

```
$ cge-cli fe --security example.ini
```

This will start the server with Shiro configured according to the given file.

⚠ **CAUTION:** In the event that the configuration is invalid the server will fail to start.

User Authorization

Authentication does not imply authorization. Regardless of what Shiro authentication realm is chosen, individual users must still be authorized to access the database. Authorizing users to access the database via the SPARQL server is a little different from authorizing them to access the database directly. In this scenario, the SPARQL server will be running as the user who launched the process, therefore all requests to the database will use that users key pair. As a result that user will need to have their key pair authorised for use by other users, as shown in the following example:

```
$ cat ~/.ssh/id_rsa.pub | sed 's/my-name/other-user/'>> /my/db/authorized_keys
```

In this example the user is authorizing their public key to be used by `other-user`. This does not grant that user the ability to connect to the database directly with this key as they would not have access to the corresponding private key. Essentially, the user delegates the ability for a process owned by themselves to use a key pair owned by themselves, on behalf of another user.

Login Mechanisms

Apache Shiro supports two login mechanisms which can be used as desired. Firstly it supports HTTP Basic authentication, in this mode any attempt to access the server that requires authentication Will send a HTTP authentication challenge back to the client. In a web browser this will typically result in the browser presenting a login prompt to the user. When the user enters their credentials this is submitted back to server for the server to verify against the configured authentication realm. In this mode every request to the server requires credentials to be presented, however most browsers will remember credentials for the life of the browser session and automatically submit them with subsequent requests.

Secondly it supports forms authentication, in this mode any attempts to access the server that requires authentication will redirect the user to the login page. The server provides a login page at `/login` so Shiro configurations should use that as the login URL. The user can then enter their credentials in a form in the browser before submitting them back to the server for the verification. In this mode the server will use cookies to identify the user, it checks the cookie against its record of logged in users to determine if the user has previously authenticated. This means that the user need only present their credentials once and thereafter need only present the cookie.

Which login mechanism is selected to be sued will depend on how the user intends to use the server. If you primarily use the server to provide SPARQL endpoints for access by SPARQL tools and libraries then you are better off using basic authentication as many tools and libraries do not support forms authentication. On the other hand if you are primarily using the server for the browser interface then forms authentication is more user-friendly.

> **NOTE:** With both mechanisms, credentials are sent unencrypted to the server and therefore are subject to interception by a malicious user/application. In order to ensure secure deployments, use the user authentication features in conjunction with the SSL features.

LDAP Integration

Apache Shiro can be configured to integrate with the system LDAP server or a central LDAP server as desired. An example configuration for this can be obtained using the `cge-cli generate shiro` command, as shown in the following example:

```
$ cge-cli generate shiro ldap > example.ini
```

Here we output the template to the file example.ini which will look something like the following:

```
[main]
# Define a LDAP realm
ldapRealm = org.apache.shiro.realm.ldap.JndiLdapRealm

# Configure the template for User lookups
# You will need to ask a system administrator what the format should be here
# The following is the default on Urika-GX systems as shipped but your system
# administrator may have
ldapRealm.userDnTemplate = uid={0},ou=People,ou=external,dc=local

# Configure to point to LDAP server of choice
# The LDAP server resides on the login1 node on Urika-GX systems as shipped
# 389 is the normal default port for LDAP servers
ldapRealm.contextFactory.url = ldap://host-login1:389

# Only uncomment and change this if your server needs a specific auth mechanism.
# By default the client should negotiate this automatically with the server
#ldapRealm.contextFactory.authenticationMechanism = DIGEST-MD5

# If your LDAP server needs credentials to access it set them here
# In most cases this should be unecessary
#ldapRealm.contextFactory.systemUsername = ldap-admin
#ldapRealm.contextFactory.systemPassword = ldap-admin-password

# Associate the realm with the security manager
securityManager.realms = $ldapRealm

# Enable auth caching, reduces load on the LDAP server
# Comment this out to disable caching
cacheManager = org.apache.shiro.cache.MemoryConstrainedCacheManager
securityManager.cacheManager = $cacheManager

# Configure the login page, /login is the page provided by the CGE SPARQL Server
authc.loginUrl = /login

[urls]
# Enable logouts
/logout = logout

# Require authentication for all paths, comment this out and uncomment the
subsequent line
# if you prefer to use HTTP Basic Authentication rather than Forms Authentication
/** = authc
#/** = authcBasic
```

On most LDAP servers, the user will only need to change two lines. Firstly the user will need to set the URL for the server:

```
ldapRealm.contextFactory.url = ldap://host-login1:389
```

To use the system LDAP server provided on a Urika-GX System this should be set to the login1 node of the system. For example if your system was named `machine` then the URL should be `ldap://machine-login1:389`. If it is required to use a central LDAP server, contact the IT department to determine the correct URL to use.

The other setting that you will need to change is the search template which is used to build the full LDAP distinguished name for a user:

```
ldapRealm.userDnTemplate = uid={0},ou=People,ou=external,dc=local
```

The example given here is the distinguished name format used by default on Urika-GX Systems. However your system administrator and/or IT department may use a very different format. If this setting is incorrect, all attempts to authenticate will fail. Please contact the system administrator and/or IT department to determine the correct distinguished name format.

There are a variety of other LDAP related settings seen in the generated template but these are all commented out as they should not be needed for most common LDAP setups. If the two aforementioned settings are correctly configured and you are still unable to authenticate successfully please contact the System administrator and/or IT department to enquire whether any advanced settings are needed.

Configuration properties

Once the preferred configuration has been put in place, it may be helpful to specify all the relevant options directly in the `cge.properties` file, instead of having to remember all the command line options. Doing this will help specify a default configuration, which is automatically picked up and applied. The following table details all the available properties that can be used to change the configuration of the SPARQL server.

Command Line option	Property	Description
`--server-host`	`cge.cli.server.host`	Sets the hostname upon which the server listens for requests
`--server-port`	`cge.cli.server.port`	Sets the port number upon which the server listens for requests
`--security`	`cge.cli.server.security`	Sets the Apache Shiro configuration file used to configure user authentication
`--ssl`	`cge.cli.server.ssl.enabled`	Enables SSL when set to `true` enable SSL
`--ssl-lax`	`cge.cli.server.ssl.lax`	When set to true, permit SSL protocols and ciphers known to be insecure
`--keystore`	`cge.cli.server.ssl.keystore`	Sets the location of the Java key store file that contains the certificate to use for SSL
	`cge.cli.server.ssl.password`	Sets the password used to access the Java key store
	`cge.cli.server.ssl.key-password`	Sets the password used to access the SSL certificate within the Java key store

7.5 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 required 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 a public / private key pair has been configured.

> **NOTE:** The following shows only key files (there will probably be 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, there may be 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, an SSH key will need to be generated . 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 it is not recommended to specify a pass-phrase when creating a key.

3. Place the public authentication key in the `.ssh/authorized_keys` file. This will enable interacting with CGE query engines started by the user on this machine (it does not allow other users to use the user's 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 have the effect of logging on to the same host the the user is currently logged on to:

```
$ 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. The user will also be prompted like this the first time the user tries 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 authenticity of the host / port pair has been verified, this pair will be added automatically to your list of known hosts and the user should not need to do this again. To avoid the need for performing the interactive Host Key verification step, see *Eliminate 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 this has been set up, it is required to authenticate the `localhost / <port number>` pairs for all 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 in the following example:

```
$ 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
```

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

7.5.1 Eliminate 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.

7.6 Grant Other Users Access to Their CGE Query Engine

The Cray Graph Engine (CGE) can protect the contents of user-owned data sets from view/modification by unauthorized users via CGE instances that you run. Regardless of this protection, it is required to protect the raw data in user-owned data sets using traditional Linux file protection, otherwise users who have access to their data can start their query engine, using their data without knowledge. To ensure that only authorized users gain access to user-owned 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, it is possible to control the list of users to whom access is granted. There are two modes of granting access to other users:

- Access to a single data set
- Access to any provided data set

A key first step to any of this is protecting owned 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 it is required to control access to owned 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 other users can be allowed to run their own instances of CGE using user-owned data, these permissions may be set any way desired.

Assuming data sets have been protected against other users, now individual users can be granted access. 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). They need to be appended to an appropriate `authorized_keys` file.

For more information, see *Configure the ACL File User Permissions* on page 86.

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

Ask users to do the following after granting 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 a user should NEVER add another user's public key to the user-owned `$HOME/.ssh/authorized_keys` file. Doing so will allow the user to login as the user who owns that file.

In the following example, it is assumed that `/lus/scratch/username/lubm0` directory contains one of user-owned 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 the user starts. Other users wanting access will need to connect with a client and will be subject to client authentication.

7.6.1 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 `authorized_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.

7.6.2 Grant Other Users Access to All of the Owned Data Sets

If it is not required to restrict access to specific data sets to a particular user, it is simpler to grant that user access to all the data sets in one authorized_keys file. CGE uses a directory located at $HOME/.cge that allows setting up configuration files that apply to all the data sets. Users can grant access to all of their data sets by creating an authorized_keys file in this directory and putting authorized public keys in that file, 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/
X72f70YL8neITsR5gotXCIZh9V0G9ar8mNDlkoshN7Jp1qiRrQjYNy93hs9BBCz9kA5V9PhGC59qypEhNovYRo48lsUvTmHK0RWOVLfIZKNCkLVmbQubmEzM
0FfUoY/
ifNbTfrV4yGH2PNA4k= my_friend@myhost
```

While this example shows placement of the global authorized_keys file in $HOME/.cge, it can be used to place the authorized_keys file in any directory. If $HOME/.cge is not a convenient place to put the authorized_keys file, follow the above procedure to place it in some other (suitably protected) directory, then use the --configDir option to cge-launch or the $CGE_CONFIG_DIR_NAME environment variable to point to that directory. If a global authorized_keys file needs to be stored on HDFS, create the file using this procedure, then copy it onto HDFS in the location of choice (appropriately protecting it). Then use an HDFS URL as the value of $CGE_CONFIG_DIR_NAME or the argument to the --configDir option to cge-launch to select that directory instead of $HOME/.cge.

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

8 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:

- **BadRank**: Assigns a "badness" score to all vertices in the graph based on their nearness to known bad vertices.
- **Betweenness Centrality**: Ranks each vertex by how frequently it is on the shortest path between vertices.
- **Page Rank**: Measures the relative importance of a vertex in a graph.
- **Community Detection Label Propagation (LP)**: Detects communities in networks and assigns vertices in the graph to communities.
- **Community Detection Parallel Louvain Method (PLM)**: Detects communities in networks and assigns vertices in the graph to communities. This method is a distributed memory implementation using CoarrayC+ + and is inspired by the shared-memory Parallel Louvain Method in NetworKit.
- **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.
- **Triangle Counting**: Counts the total number of triangles in a graph.
- **Triangle Finding**: Finds all the triangles in the graph.
- **Vertex Triangle Counting**: Gathers statistics on the vertices based on the triangles they participate in and for non-cyclic triangles, their position in the triangle.

8.1 Combine 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.

8.2 Invocation of a Graph Function

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

- CONSTRUCT
- INVOKE
- PRODUCING
- SELECT-WHERE

8.2.1 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.

8.2.2 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.

8.2.3 The PRODUCING Clause

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 variables that will 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
…
```

8.3 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, i.e, 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.

8.4 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
```

8.5 Bad Rank

URI

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

Description

The Bad Rank algorithm assigns a "badness" score to all vertices in the graph based on their nearness to known bad vertices.

Inputs and Default Values

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. 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.

8.6 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.

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

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	`1`

Parameter	Description	Default Value
	difference between the maximum and minimum betweenness scores found among all the vertices. Normalized scores will be between `0.0` and `1.0`.	

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)
```

Special Consideration for Graphs with Very Large Diameter

The value of the `cge.server.BCmaxActiveLevels` NVP parameter can be used to better handle graphs with large diameters. The default setting for this parameter is 100 and can be increased if needed.

If the value of this parameter is set to a value that is too low, the database will remain up and running, but the query will be halted, and the system will return a message indicating that the `cge.server.BCmaxActiveLevels` parameter's value (i.e. the allocation size) needs to be increased.

The error message is written to a CGE log file as well as to the front end.

The message written to the CGE log file will be similar to the following:

```
Warning, graph diameter is larger than current allocation for LevelSet data
structure.
Use NVP parameter BCmaxActiveLevels to increase the size of the allocation
currently set to X levels.
```

Here *X* is used as an example for the current value of the `BCmaxActiveLevels` parameter.

Similarly, the following message will be returned to the CGE front end:

```
graph diameter is larger than current allocation for LevelSet data structure.
Use NVP parameter BCmaxActiveLevels to increase the size of the allocation
```

8.7 Community Detection Label Propagation (LP)

URI

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

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.community_detection_LP(5)
PRODUCING ?vertex ?comm
}
ORDER BY ?comm
```

8.8 Community Detection Parallel Louvain Method (PLM)

URI

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

Description

The Parallel Louvain Method is used for detecting communities in networks and assigns vertices in the graph to communities. The `community_dection_PLM` method is a distributed memory implementation using CoarrayC++ and is inspired by the shared-memory Parallel Louvain Method in NetworKit, an open-source package (https://networkit.iti.kit.edu), and corresponding paper “*Engineering Parallel Algorithms for Community Detection in Massive Networks*” by Christian L. Staudt and Henning Meyerhenke. The algorithm can take up to two input parameters. The first parameter controls the maximum number of PLM steps taken. The second parameter is number of initial Label Propagation steps to take to initialize the starting communities before running the PLM steps. If the number of Label Propagation steps is set to 0, each vertex is initially assigned to its own community.

If no vertices are moved during a PLM step, the routine will exit early, returning the community assignments corresponding to the largest computed modularity score found up to this point.

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
Maximum number of PLM steps. An early exit is included if convergence is detected (if no vertices are moved during a PLM step, the process has converged).	`20`
Number of Label Propagation steps to be run to initialize the starting community assignments prior to running the PLM steps.	(Input number of PLM steps)/2

Outputs

A call to the 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: Parallel Louvain

```
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.community_detection_PLM(25,5)
PRODUCING ?vertex ?comm
}
ORDER BY ?comm
```

8.9 Page Rank

URI

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

Description

Page Rank can be used to measure the relative importance of a vertex in a graph.

Inputs and Default Values

Outputs

Page 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 Page Rank value of the vertex.

Example: Page Rank

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

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

  SELECT ?vertices ?pagerank
  WHERE {
    CONSTRUCT{
      ?sub ?pred ?obj .
    }
    WHERE{
      { ?sub ?pred ?obj . }
    }
    INVOKE cray:graphAlgorithm.pagerank(0.0005,0.85)
    PRODUCING ?vertices ?pagerank
  }
  ORDER BY DESC(?pagerank)
```

8.10 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 (Source Target) 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
}
```

8.11 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 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
}
```

8.12 Triangle Counting

URI

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

Description

Triangle Counting is used to count the total number of triangles in a graph.

Inputs and Default Values

- **Vector inputs** - None.
- **Scalar inputs** - This algorithm accepts a single integer scalar argument. The value of this integer ranges from 0 to 4 and specifies which types of triangles are to be included in the count.
  - `0`: Return a count of all the triangles in the graph, both cyclic (including rotations) and non-cyclic triangles
  - 1: Return a count of all the unique triangles in the graph, both cyclic and non-cyclic triangles
  - 2: Return a count of only the non-cyclic triangles
  - 3: Return a count of only the cyclic triangles (including rotations)
  - 4: Return a count of only the unique cyclic triangles

Outputs

This algorithm returns a single integer containing the number of triangles.

Example: Triangle Counting

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

  SELECT ?total_num_triangles
  WHERE {
    CONSTRUCT{
      ?sub ?pred ?obj .
    }
    WHERE{
      ?sub ?pred ?obj .
    }
    INVOKE cray:graphAlgorithm.triangle_counting(1)
    PRODUCING ?total_num_triangles
```

8.13 Vertex Triangle Counting

URI

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

Description

The Vertex Triangle Counting algorithm is used to gather statistics on the vertices based on the triangles they participate in and for non-cyclic triangles, their position in the triangle.

Inputs and Default Values

- **Vector inputs**- None.
- **Scalar inputs** - This algorithm accepts a single integer scalar argument. The value of this integer ranges from 0 to 4 and specifies which types of triangles are to be included in the counting statistics.
  - `0`: Return a count of all the triangles in the graph, both cyclic (including rotations) and non-cyclic triangles.
  - 1: Return a count of all the unique triangles in the graph, both cyclic and non-cyclic triangles
  - 2: Return a count of only the non-cyclic triangles
  - 3: Return a count of only the cyclic triangles (including rotations)
  - 4: Return a count of only the unique cyclic triangles

Outputs

Output is a four-column intermediate result. Each row in the intermediate results contains a vertex URI followed by a total count of the triangles for which it participates as either a `through_vertex`, `in_vertex`, or `out_vertex`, respectively. The `PRODUCING` clause should be interpreted as "`vertexID`","`through_count`", "`in_count`", "`out_count`", where the counts refer to the number of triangles in which the vertex participates in that role.

Example: Vertex Triangle Counting

```
PREFIX cray: <http://cray.com/>
  SELECT ?id ?through ?in ?out
  WHERE {
    CONSTRUCT{
      ?sub ?pred ?obj .
    }
    WHERE{
      ?sub ?pred ?obj .
    }
    INVOKE cray:graphAlgorithm.vertex_triangle_counting(0)
    PRODUCING ?id ?through ?in ?out
  }
```

8.14 Triangle Finding

URI

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

Description

The Triangle Finding algorithm is used to find all the triangles in the graph. The output can be customized to return either all triangles, or only the cyclic or non-cyclic triangles. The number of triangles in a given region of a graph is a good indicator of the density of that part of the graph.

Inputs and Default Values

- **Vector inputs**- None.
- **Scalar inputs** - This algorithm accepts a single integer scalar argument. The value of this integer ranges from 0 to 4 and specifies which types of triangles are to be output..
  - `0` - Return all the triangles in the graph, both cyclic (including rotations) and non-cyclic triangles
  - 1: Return all the unique triangles in the graph, both cyclic and non-cyclic triangles
  - 2: Return only the non-cyclic triangles
  - 3: Return only the cyclic triangles (including rotations)
  - 4: Return only the unique cyclic triangles

Outputs

The code returns a four-column IRA . Each row in the IRA represents the three URIs of the vertices of a triangle followed by a cyclic flag (set to `1` for cyclic, `0` for non-cyclic). The non-cyclic triangles are written out in the order of through_vertex, in_vertex, out_vertex. The cyclic flag is considered optional in the `PRODUCING` clause in the case where only the URIs of the vertices are needed.

Example: Triangle Finding

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

  SELECT ?vertexID1 ?vertexID2 ?vertexID3 ?cyc
  WHERE {
    CONSTRUCT{
      ?sub ?pred ?obj .
    }
    WHERE{
      ?sub ?pred ?obj .
    }
    INVOKE cray:graphAlgorithm.triangle_finding(1)
    PRODUCING ?vertexID1 ?vertexID2 ?vertexID3 ?cyc
```

9 CGE Extension Functions

CGE provides a number of extension functions, including:

- Interval analytics functions.
- Haversine functions.
- Square root function.

9.1 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.

CGE interval analytic 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

CGE interval functions are case-sensitive and work with any type that has a < comparison, e.g., numerics and strings.

Table 16. 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.

Function	Description
`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. ● `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.

The arguments provided to the `listmin()`, `listmax()`, `iscontinuous()` and `isintersecting()` functions 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

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.

9.2 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#>
```

9.3 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 17. 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.

9.4 Custom Aggregate Functions

CGE supports the following custom aggregate functions:

- `variance`
- `standard deviation`
- `geometric mean`
- `mode`
- `median`

Table 18. Custom Aggregate Functions

Function	Purpose
`variance`	Returns the variance of an expression.
`standard deviation`	Calculates the standard deviation of a set of numeric values. Requires at least two values.
`geometric mean`	Calculates the *n*th root of the product of the numbers, where *n* is the count of numbers.
`mode`	Returns the most frequently occurring number in a group of supplied arguments.
`median`	Calculate the median, which is the value separating the higher half of a data sample, a population, or a probability distribution, from the lower half.

Examples

- `variance`

```
SELECT ?p (AGG<http://www.dotnetrdf.org/leviathan#variance>(?o) AS ?RESULT)
WHERE
  { ?s ?p ?o}
GROUP BY ?p
```

- `geometric mean`

```
SELECT ?p (AGG<http://www.dotnetrdf.org/leviathan#geometric_mean>(?o) AS ?
RESULT)
WHERE
  { ?s ?p ?o}
GROUP BY ?p
```

- `standard deviation`

```
SELECT ?p (AGG<http://www.dotnetrdf.org/leviathan#standard_deviation>(?o) AS ?
RESULT)
WHERE
  { ?s ?p ?o}
GROUP BY ?p
```

- mode

  ```
  SELECT ?p (AGG<http://www.dotnetrdf.org/leviathan#mode>(?o) AS ?RESULT)
  WHERE
    { ?s ?p ?o}
  GROUP BY ?p
  ```

- median

  ```
  SELECT ?p (AGG<http://www.dotnetrdf.org/leviathan#median>(?o) AS ?RESULT)
  WHERE
    { ?s ?p ?o}
  GROUP BY ?p
  ```

Custom aggregate functions can be freely mixed together or with standard SPARQL aggregate functions, such as:

- SUM
- MIN
- MAX
- SAMPLE
- AVG
- GROUPCONCAT

For example:

```
SELECT ?p
    (AGG<http://www.dotnetrdf.org/leviathan#variance>(?o) AS ?RESULT1)

   (SUM (?o) AS ?RESULT2)
    (AGG<http://www.dotnetrdf.org/leviathan#median>(?o) AS ?RESULT3)
```

CAUTION: The DISTINCT flavors of custom aggregates are currently not supported.

Geometric Mean

Geometric mean is defined as the *n*th root of the product of n values. The product's absolute value is used under the radical sign to avoid negative numbers. The result in this case will be zero. The product's absolute value is used under the radical sign to avoid negative numbers. In other words, if the product happens to be negative, that value is negated to make it positive and then its root is retrieved.

10 Cray Graph Engine (CGE) Property Path Support

CGE does not natively support all the SPARQL 1.1 property paths features, 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 19. 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

Operator	Example	Description	Additional Notes
			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 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.

11 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 allows re-logging into the login node without a password, then the SSH keys are set up sufficiently for using CGE.

```
$ ssh localhost
```

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:

⚠ **CAUTION:** 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 83. 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.

⚠ **CAUTION:** 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 at least one of the following if the data set is being built for the first time with CGE (only one of these will actually be used):

  - `dataset.nt` - This file contains triples and must be named dataset.nt
  - `dataset.nq` - This file contains quads and must be named dataset.nq
  - `graph.info` - This file contains a list of pathnames or URLs to files containing triples or quads and must be named `graph.info`.
- `-l` - Specifies a log file to capture the command output from the run. 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. If the `-l` option is specified, the `cge-launch` command runs silently, producing no output to the terminal `stdout`/`stderr`.

For more information, see *Launch the CGE Server Using the cge-launch Command* on page 22 and *The CGE Database Build Process* on page 13.

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 *CGE CLI* 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 fe --ping
```

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. 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 *Launch the CGE Web Server Using the fe Command* on page 39.

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 *CGE GUI* on page 52.

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 *Shutdown the CGE Server Using the shutdown Command* on page 46.
- Shut down the CGE front end if it was started.

Additional Information

Cancelling a query - To cancel a query, hit CTRL-C on the window where the CGE server was launched or locate the CGE server instance's PID on the login node and use `kill -INT <PID>`. After that, re-launch CGE.

12 Get Started with Using CGE

Prerequisites

This procedure requires CGE to be installed on the system.

About this task

This procedure can be used to get started with using CGE and can be considered as a "Hello World" program. In this procedure, a simple query is executed on a small RDF triples database. This procedure provides instructions for executing queries and viewing the results via the CGE CLI and the front end.

Use the `cge-cli help` command to view a full range of CGE CLI commands. Use the `–h` option of any command to view detailed help information about any specific command.

For a full set of CGE features, built in functions, graph algorithms, CGE API, troubleshooting and logging information, review the Cray Graph Engine (CGE) Users guide at *https://pubs.cray.com*.

Procedure

Authentication Setup

1. Set up SSH keys.

```
$ ssh localhost
```

If the preceding command allows re-logging into the login node without a password, then the SSH keys are set up sufficiently for using CGE. 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 text 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. On the other hand, if the aforementioned text fails, there are no SSH keys defined yet. The following commands can be used to set them up.

⚠ **CAUTION:** Before executing the following commands, 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
```

Dataset Creation

2. Create a file named `dataset.nt` and store it in a directory that has been selected or created for it.

This directory must be a new directory and contain at least one of the following if the data set is being built for the first time with CGE (only one of these will actually be used):

- `dataset.nt` - This file contains triples and must be named dataset.nt
- `dataset.nq` - This file contains quads and must be named dataset.nq
- `graph.info` - This file contains a list of pathnames or URLs to files containing triples or quads and must be named `graph.info`.

This is the original, human readable representation of the database. The following example data, which should be added to `dataset.nt`, can be used for this procedure.

```
<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"  .
```

Results Directory Creation and CGE Server Start-up

3. Load the CGE module.

```
$ module load cge
```

4. 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.

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
```

Query Execution via CGE CLI

5. Execute a query using the CGE CLI.

```
$ cge-cli query example.rq
0 [main] WARN com.cray.cge.cli.CgeCli  - User data hiding is enabled, logs will obscure/omit user
data.  Set cge.server.RevealUserDataInLogs=1 in the in-scope cge.properties file to disable this
behaviour.
5 [main] INFO com.cray.cge.cli.commands.queries.QueryCommand  - Received 1 queries to execute
13 [main] INFO com.cray.cge.cli.commands.queries.QueryCommand  - Running Query 1 of 1
0              6              123         0              file:///mnt/central/user/results/
queryResults.2017-07-04T13.59.57Z000.18232.tsv
688 [main] INFO com.cray.cge.cli.commands.queries.QueryCommand  - Query 1 of 1 succeeded
```

In the preceding example, the `example.rq` file contains the following query:

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

Use the following query to print just "Hello World" as the output:

```
SELECT ?greeting ?object
WHERE
{
  <http://cray.com/example/greeting> <http://cray.com/example/text> ?greeting .
  <http://cray.com/example/spaceObject> <http://cray.com/example/hasName> ?object .
  FILTER(?greeting = "Hello" && ?object = "World")
}
```

Results Review

6. List the contents of the results directory and review the contents of the output file to verify that the query's results are stored in the output directory specified in the `cge-launch` command.

```
$ 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”
```

CGE Front End Launch

7. Launch the CGE front end in another terminal window.

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

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

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 &
```

8. Point a browser at `http://`*`loginNode`*`:3756` to launch web UI, where *`loginNode`* is the name of the login node the front end is launched from.

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

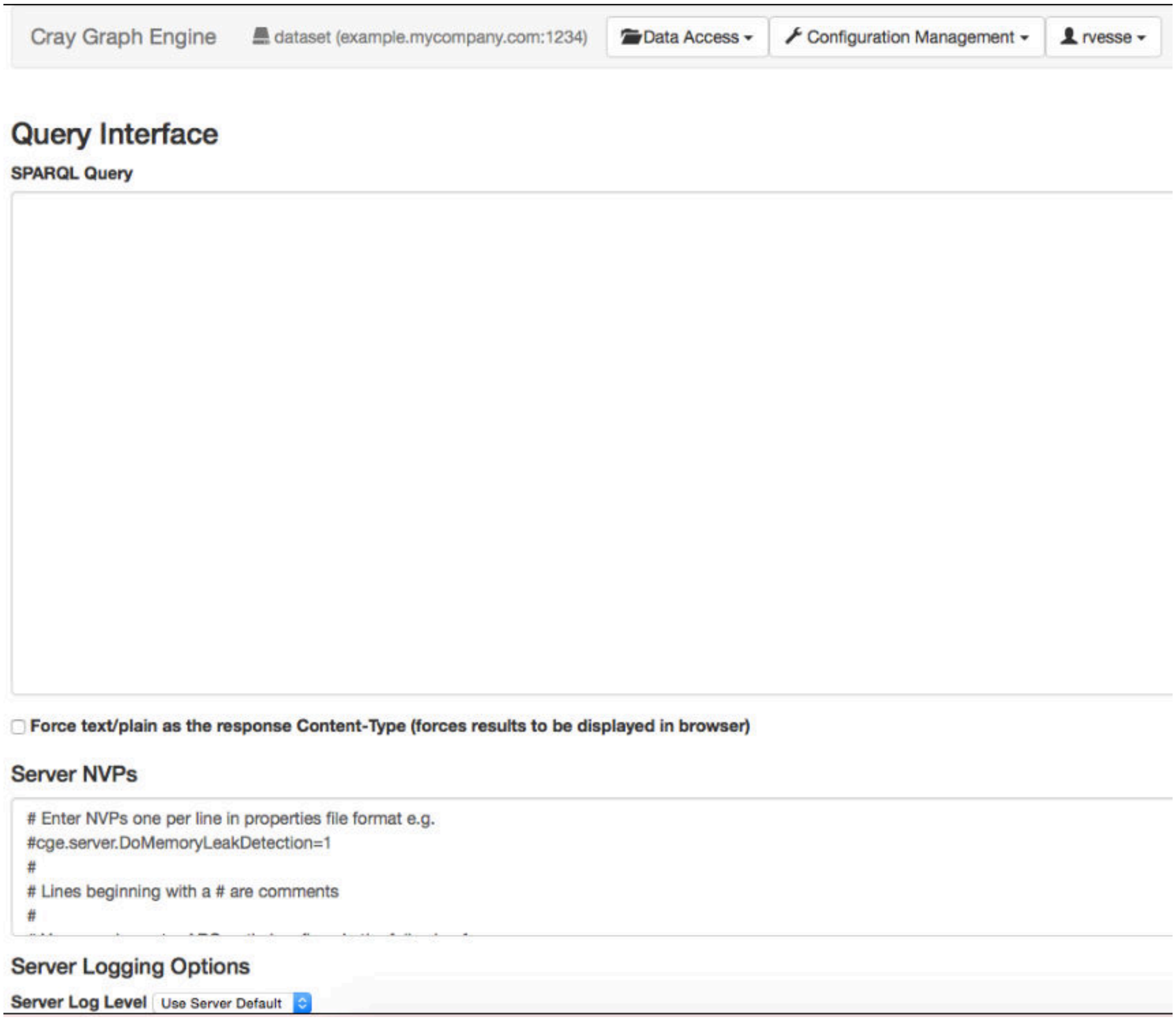
When the CGE front end has been launched, a message similar to the following will be returned on the command-line:

```
49 [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
```

Query Execution via the CGE Front End

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

Figure 21. CGE Query 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.
#cge.server.DoMemoryLeakDetection=1
#
# Lines beginning with a # are comments
#
```

Server Logging Options

Server Log Level: Use Server Default

The following example query will match the data and example output shown in the next step:

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

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.

CGE Front End Termination

10. Quit the terminal using the `CTRL+C` keyboard shortcut.

CGE Server Shutdown

11. Execute the following command to halt the CGE server, if needed.

```
$ cge-cli shutdown
```

13 Support for Simple GraphML Files

CGE enables importing simple GraphML files and generating the corresponding quads for the given graph(s). To enable importing a GraphML file, the user can either list a GraphML file in a `graph.info` file as part of a database build, or load a GraphML file. When CGE processes an input file, any file that ends with the `.graphml` extension will be treated as a GraphML file.

The syntax supported for GraphML files is based on the DTD specification provided at: *http://graphml.graphdrawing.org/*

The following is a sample GraphML file that represents a simple graph:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4"/>
    <edge id="e1" source="n0" target="n2"/>
    <edge id="e2" source="n1" target="n2"/>
    <edge id="e3" source="n2" target="n3"/>
    <edge id="e4" source="n3" target="n4"/>
  </graph>
</graphml>
```

Limitations

There are multiple limitations in the current support for GraphML files, including the following:

- The `xml` and `graphml` elements are parsed, but otherwise ignored.
- Edge data is currently ignored.
- Default edge direction for a graph is ignored.
- Edge direction attribute is ignored.
- Default values for data are ignored.
- Elements in a graph are limited to descriptions, data, nodes and edges.
- Nodes and edges can only contain descriptions or data as subelements.
- Nested graphs are not supported.

CGE will report warning or error messages to the log file for any incorrect syntax or unsupported features.

When translating an edge to a quad, CGE will convert the edge identifier as well as the source and target identifiers to URIs.

For example, given the following edge from the example above:

```
<edge id="e1" source="n0" target="n2"/>
```

CGE would generate the following quad:

```
<urn:n0> <urn:e2> <urn:n2> <urn:G> .
```

Note that when converting the identifier to a URI, CGE will insert the urn: prefix by default. Also, if any error is found when parsing an edge no quad will be generated for that edge. For example, if a node referred to by an edge does not exist in the given graph, or if there was an error when parsing the node declaration, these errors will prevent a quad from being generated for an edge.

NVPs for GraphML Support

- `cge.server.ExportGMLRDFEnable` - Setting this NVP to 1 will cause CGE to export the quads generated for a given GraphML file to an nt file of the same name as the input GraphML file but with the `nt` extension. For example, if a `graph.info` file includes the line:

  ```
  /my/path/to/file_name.graphml
  ```

  The given NVP is set to 1 then CGE will write the quads produced by the GraphML file to an `nt` file named:

  ```
  /my/path/to/file_name.nt
  ```

  Exporting the quads to an nt file can be useful if the quads will be loaded multiple times since loading quads is faster and uses less memory than loading from a GraphML file. This NVP is off by default.

- `cge.server.GMLInsertPrefix` - Setting this to 1 will cause CGE to insert the urn: prefix when converting identifiers for graphs, nodes, and edges to URIs. For example, the following edge:

  ```
  <edge id="e1" source="n0" target="n2"/>
  ```

  would result in URIs of `<urn:e1>, <urn:n0> and <urn:n2>` for the edge, source and target identifiers, respectively. This NVP is on by default.

- `cge.server.GMLCheckPrefix` - Setting this to `1` will cause CGE to check an identifier for a known prefix before inserting the urn: default prefix. The prefixes that CGE will check for are:

  - `urn:`
  - `http:`
  - `https:`

  If a graph, node or edge identifier starts with one of these prefixes and this NVP is set, CGE will not insert the `urn:` prefix. For example, given the following edge:

  ```
  <edge id="http://www.mysite.com/e1" source="n0" target="n2"/>
  ```

  and having this NVP set will result in the following URIs:

  - `<http://www.mysite.com/e1>`

- `<urn:n0>`
- `<urn:n2>`

Notice that since the source and target identifiers did not include a known prefix, CGE will insert the `urn:` prefix by default.

14 Lustre Striping on CGE

Striping a Lustre directory can help achieve better parallel I/O performance. When loading input NT/NQ files into CGE from Lustre, it is important that the files use the optimal striping settings for the given Lustre file system. For larger files, load performance can be improved if the input dataset(s) have a larger stripe count. Use the `lfs getstripe` command to determine the striping information for a directory and the files contained within it.

For example, the following will get the striping information for the current directory:

```
$ lfs getstripe .
```

The striping information for a file must be set before it is created. An easy way to do this is to set the striping information on a directory and then copy the files into that directory because files will inherit striping information from their parent directory. For example, the following command can be used to set the stripe count to 16 and the stripe size to 16 MB on the current directory:

```
$ lfs setstripe -c 16 -S 16m .
```

Striping is also important for performance when writing a large file to Lustre. For CGE, directory striping can significantly improve performance when writing a compiled database to Lustre, checkpointing a database, or writing a large results file from a query.

⚠ **CAUTION:** It may not always be possible to change the striping on a directory that contains (or has contained) files. Therefore, it is safer to set striping on a newly created directory.

15 CGE API

15.1 CGE API Versioning

CGE API file versioning consists of filename subtext "v*X.Y.Z*", where *X*, *Y*, and *Z* represent integers. The following table correlates versions of CGE with API version numbers:

CGE	Java API	Python API
2.0UP00	v1.0.0	none
2.5UP00	v1.1.0	v1.0.0
3.0UP00	v1.3.0	v1.0.1
3.1UP00	v1.4.1	v1.0.1

The CGE user guide sections describing each type of API use v*X.Y.Z* for file versioning. Users are expected to replace this with the appropriate version of interest from the table.

15.2 Prepare the Environment for Using CGE Java API on Urika-XC

Prerequisites

This procedure requires CGE to be installed on the Urika-XC system.

About this task

When using the CGE Java API on Urika-XC systems,certain settings need to be put in place before using the CGE Java API on Urika-XC systems.

Procedure

1. Load the analytics module and allocate resources.

   In the following example, *X* is the number of compute nodes to utilize.

   ```
   $ module load analytics
   $ salloc -N X start_analytics
   ```

2. Copy the CGE Java API JAR file from the installed directory to the user's local directory.

   ```
   $ cp /opt/cray/cge/default/lib/java/cge-java-api-vX.Y.Z-with-dependencies.jar\
   /home/users/$USER
   ```

   All references to this JAR file in this publication must be made to the users local copy.

15.3 CGE Java API

This feature is currently supported fully on Urika-GX and partially on Urika-XC. Specifically, launching CGE is currently only supported on Urika-GX, whereas all other CGE Java API features are supported on Urika-XC.

The CGE Java API consists of four Java JAR files and a ReadMe file:

1. `cge-java-api-vX.Y.Z-with-dependencies.jar` - Contains executable Java classes. Using the executable Java classes, users can write their own Java code to perform CGE actions like launching the server, querying, updating, checkpointing, etc.
2. `cge-java-api-examples-vX.Y.Z-sources.jar` - Contains sample source code. Users can build their own Java programs referring to these examples.
3. `cge-java-api-vX.Y.Z-javadocs.jar` - Contains documentation of Java classes that the CGE API is comprised of.
4. `cge-java-examples-vX.Y.Z-with-dependencies.jar` - Contains executable sample source code. Samples provided with the CGE API can also be executed, as their entry points are included in the executable sample source code
5. `ReadMe_JavaAPI.txt` - Contains commands and scripts documented in this guide. These commands and scripts are suitable for copying and pasting for execution.

Where v*X.Y.Z* is the version code documented in *CGE API Versioning* on page 144.

This section describes how to utilize these `.jar` files.

Source Code Extraction

To extract the Java source code, run the following command:

```
$ jar -xf PATH_TO_JAR/cge-java-api-examples-vX.Y.Z-sources.jar
```

A directory structure similar to the following should appear:

```
./com/cray/cge/api/examples/hooks/InheritIOHook.java
./com/cray/cge/api/examples/Checkpoint.java
./com/cray/cge/api/examples/CheckpointExisting.java
./com/cray/cge/api/examples/ComprehensiveExample.java
./com/cray/cge/api/examples/Config.java
./com/cray/cge/api/examples/ExampleUtils.java
./com/cray/cge/api/examples/IsRunning.java
./com/cray/cge/api/examples/LaunchAndShutdown.java
./com/cray/cge/api/examples/LaunchAndTerminateOnJvmExit.java
./com/cray/cge/api/examples/LaunchOnly.java
./com/cray/cge/api/examples/Query.java
./com/cray/cge/api/examples/QueryExisting.java
./com/cray/cge/api/examples/Shutdown.java
./com/cray/cge/api/examples/Status.java
```

```
./com/cray/cge/api/examples/StatusExisting.java
./com/cray/cge/api/examples/Update.java
./com/cray/cge/api/examples/UpdateExisting.java
./com/cray/cge/api/examples/QueryWithNVP.java
```

The `.java` files are source-code examples that users can refer to when building their own Java programs. These are also included in the executable Java classes, so users can execute them directly if desired.

Extraction of API Class Documentation

To extract the API class documentation, run the following command:

```
$ jar -xf PATH_TO_JAR/cge-java-api-vX.Y.Z-javadocs.jar
```

A directory structure similar to the following should appear. Please note that there is an extensive directory structure that exists under the `./com` directory, but it is not displayed in the following.

```
./META-INF
./META-INF/MANIFEST.MF
./resources
./resources/titlebar.gif
./resources/tab.gif
./resources/titlebar_end.gif
./resources/background.gif
./com
./allclasses-frame.html
./overview-frame.html
./overview-summary.html
./package-list
./deprecated-list.html
./serialized-form.html
./index.html
./help-doc.html
./index-all.html
./allclasses-noframe.html
./constant-values.html
./stylesheet.css
./overview-tree.html
```

This file system is meant to be run with a web-browser.

- To view on Windows systems, right-click on **overview-summary.html** and then select the **open with** menu option to open the file using the web-browser of interest, such as Internet Explorer or Chrome.
- To execute on Linux systems, ensure that an X-Windows server is running on the target computer, and that the `$DISPLAY` environment variable is set appropriately. Then execute with the web-browser of interest, for example using `firefox help-doc.html&`.

In either case, the documentation should appear for point-and-click viewing.

How to Use API Executables

There are several ways to utilize the executable Java classes `.jar` files, as illustrated by the following:

- Use Java API via Maven
- Use Java API via Java Development Kit (JDK)
- Use Java API via pre-built `main` entry points

- Use Java API Comprehensive Example program

15.3.1 Build CGE Java Applications Using Maven

About this task

The following procedure illustrates a use-case for using Maven to develop a Java application program to execute the `cge-launch` command. This sample program will utilize the `LaunchOnly.java` sample file.

Procedure

1. Create an application framework.

   ```
   $ mvn archetype:generate -DgroupId=com.mycompany.launchonly \
   -DartifactId=my-launchonly -DarchetypeArtifactId=maven-archetype-quickstart \
   -DinteractiveMode=false
   ```

   This is a standard Maven command to make a framework for developing a Java application; in this case, a framework is created in the new `my-launchonly` area.

2. Switch to the `my-launchonly` directory.

   ```
   $ cd my-launchonly
   ```

3. Install executable classes as a local `jar` file.

   ```
   $ mvn install:install-file \
   -Dfile=./cge-java-examples-vX.Y.Z-with-dependencies.jar  \
   -DgroupId=com.cray.cge.api \
   -DartifactId=cge-user-apis \
   -Dversion=1.0.0 \
   -Dpackaging=jar
   ```

   This is a standard Maven command for creating a directory structure for an executable `.jar` file.

4. Copy the `LaunchOnly.java` file into this area of the new `mycompany` directory.

   ```
   $ cp .../LaunchOnly.java \
   ./src/main/java/com/mycompany/launchonly/LaunchOnly.java
   ```

5. Develop a `pom.xml` file for the application or overwrite the default `pom.xml` file.

   There can be many variations of this file, the following is shown as a suggestion for the contents of this file:

   ```
   <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
     <modelVersion>4.0.0</modelVersion>
     <groupId>com.cray.cge.api.examples</groupId>
     <artifactId>my-launchonly</artifactId>
     <packaging>jar</packaging>
     <version>1.0.0</version>
     <name>my-launchonly</name>
     <url>http://maven.apache.org</url>
     <build>
       <plugins>
         <plugin>
           <groupId>org.apache.maven.plugins</groupId>
           <artifactId>maven-assembly-plugin</artifactId>
           <version>2.6</version>
             <executions>
               <execution>
   ```

```
               <goals>
                  <goal>attached</goal>
               </goals>
               <phase>package</phase>
               <configuration>
                 <descriptorRefs>
                   <descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
                <archive>
                  <manifest>
                    <mainClass>com.cray.cge.api.examples.LaunchOnly</mainClass>
                  </manifest>
                </archive>
              </configuration>
           </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.cray.cge.api.spark</groupId>
        <artifactId>cge-user-apis</artifactId>
        <version>1.0.0</version>
    </dependency>
  </dependencies>
</project>
```

6. Build the package.

Users can first delete all files in their `~/.m2` directory, as Maven will download what it needs to build the package into this area.

```
$ mvn clean package
```

7. Execute the code using one of the following:

- `$ java -jar ./target/my-launchonly-1.0.0-jar-with-dependencies.jar /path-to-dataset ./`

-
  ```
  $ java -cp ./target/my-launchonly-1.0.0-jar-with-dependencies.jar
  com.cray.cge.api.examples.LaunchOnly /path-to-dataset ./
  ```

-
  ```
  $ export CLASSPATH=./target/my-launchonly-1.0.0-jar-with-dependencies.jar;
  java com.cray.cge.api.examples.LaunchOnly /path-to-dataset ./
  ```

Note the following items:

- The `./` argument specifies where the program will place the result files.
- *`/path-to-dataset`* is the directory path to the user's dataset area, e.g. `/lus/scratch/ripple/mkdb/sp2b/25k`
- Outputs are the `cge_launcher.log` and `cge_runtime.log` files.

Outputs to `stdout` from execution should be similar to the following:

```
Launcher arguments are:
cge-launch -d /path-to-dataset -o my-launchonly/. -l cge_runtime.log -p 12345 --nodeCount 1 --imagesPerNode 1 --
sessionTimeout 900
Starting CGE...
CGE not yet ready (1 seconds elapsed)
CGE not yet ready (2 seconds elapsed)
CGE not yet ready (3 seconds elapsed)
CGE ready in 7 seconds
CGE is running
```

8. Shut down the CGE CLI.

```
$ cge-cli shutdown --db-port 22334
```

15.3.2 Build CGE Applications Using JDK

About this task

This procedure illustrates a use-case where programmers utilize JDK directly for Java program development.

Procedure

1. Create a program.

   ```
   $ mkdir jdk_is_running
   $ cd jdk_is_running
   ```

   This sample program will determine if the CGE server is running.

2. Create a directory to put the `.jar` files into and move to that directory

   ```
   $ mkdir cray_jars
   ```

3. Copy the `cge-java-examples-vX.Y.Z-with-dependencies.jar` file to this area.

   ```
   $ cp /opt/cray/cge/default/lib/java/cge-java-examples-vX.Y.Z-with-dependencies.jar cray_jars
   ```

4. Create a file called `Manifest.txt`.

   ```
   $ touch Manifest.txt
   ```

5. Edit the `Manifest.txt` file to contain the following lines.

   ```
   Main-Class: com.cray.cge.api.examples.IsRunning
   Class-Path: cray_jars/cge-java-examples-vX.Y.Z-with-dependencies.jar
   <blank_line>
   ```

   IMPORTANT: The file containing the above lines must have a blank line (press Enter/Return to create a blank line) the end of the file. in the preceding example, `<blank_line>` is used to indicate a line with no characters.

6. Make a directory to locate the source file.

   ```
   $ mkdir -p com/cray/cge/api/examples
   $ cp IsRunning.java com/cray/cge/api/examples
   ```

7. Copy the `IsRunning.java` sample code into the new directory.

   NOTE: The directory name must match the package name in the source code.

   See *Source Code Extraction* on page 145 for the location of `IsRunning.java`.

8. Build the package.

   ```
   $ javac -classpath cray_jars/cge-java-examples-vX.Y.Z-with-dependencies.jar \
   com/cray/cge/api/examples/IsRunning.java
   ```

9. Run via the Java interpreter.

```
$ java -cp cray_jars/cge-java-examples-vX.Y.Z-with-dependencies.jar: \
com/cray/cge/api/examples/IsRunning
```

The output from execution will indicate that either CGE is running or CGE is not running .

10. Build executable JAR file.

```
$ jar cvfm0 out.jar Manifest.txt com/cray/cge/api/examples/IsRunning.class
```

11. Run the executable JAR file.

```
$ java -jar out.jar
```

Output from execution should be either `CGE is running` or `CGE is not running`.

15.3.3 Build CGE Applications Using Pre-built Main Entry Points

About this task

This procedure illustrates a use-case where developers can run the `main` entry points in the `cge-java-examples-v1.0.0-with-dependencies.jar` file directly. These correspond with the example Java files containing the source line `public static void main(String[] args)`, of which there are several, for example: `./com/cray/cge/api/examples/Shutdown.java`. See *Source Code Extraction* on page 145 for the location of `Shutdown.java`.

From any directory that contains `cge-java-examples-vX.Y.Z-with-dependencies.jar`. The following examples show actual paths to datasets and repositories.

Procedure

1. Launch the CGE server

```
$ java -cp cge-java-examples-vX.Y.Z-with-dependencies.jar \
com.cray.cge.api.examples.LaunchOnly /lus/scratch/ripple/mkdb/sp2b/25k ./
```

NOTE: The `./` argument specifies where the program will place the result files. In this example, the `/lus/scratch`..." area contains a typical `sp2b` test dataset.

Output includes the files `cge_launcher.log` and `cge_runtime.log`. Outputs that appear on stdout should be similar to the following:

```
Launcher arguments are:
cge-launch -d /lus/scratch/ripple/mkdb/sp2b/25k  \
-o /ufs/home/users/$USER/my_repository/.  \
-l /ufs/home/users/$USER/my_repository/cge_runtime.log  \
-p 22334 --nodeCount 1 --imagesPerNode 1 --sessionTimeout 900
Starting CGE...
CGE not yet ready (1 seconds elapsed)
CGE not yet ready (2 seconds elapsed)
CGE ready in 5 seconds
CGE is running
```

2. Check if the CGE server is running

```
$ java -cp cge-java-examples-vX.Y.Z-with-dependencies.jar com.cray.cge.api.examples.IsRunning
```

The following will be displayed on stdout:

```
CGE is running
```

3. Execute a query

```
$ $ java -cp  cge-java-examples-vX.Y.Z-with-dependencies.jar com.cray.cge.api.examples.QueryExisting  \
/home/users/$USER/cge-benchmark/cge_queries/sp2b/2.txt > query_results.out > SELECT(COUNT(?s) as ?CNT) {?s ?p ?o}
```

In the preceding example, `2.txt` is a pre-defined query meant for the sp2b-25k dataset. It is also possible to create a `2.txt` file with `SELECT(COUNT(?s) as ?CNT) {?s ?p ?o}` as the only line.

The query results will by default go to stdout and a `.tsv` file. In this example, the query results are extensive so we redirect the default output to a file named `query_results.out`. The generated file `queryResults.2016-05-12T16.07.46Z000.12512.tsv` is also shown:

```
$ ls -l query_results.out
-rw-r--r-- 1root 292653May 1211:07query_results.out
$ ls -l queryResults.2016-05-12T16.07.46Z000.12512.tsv
-rw-r--r-- 1root 299154May 1211:07queryResults.2016-05-12T16.07.46Z000.12512.tsv
```

4. Update

```
$ java -cp  cge-java-examples-vX.Y.Z-with-dependencies.jar \
com.cray.cge.api.examples.UpdateExisting > updates.log
```

This update to the dataset is the simple default `INSERT DATA {<urn:s> <urn:p> <urn:o>}`, found in the `Update.java` sample file. See *Source Code Extraction* on page 145 for the location of `Update.java`. An argument such as `.examples.UpdateExisting ./path-to-file` can be used to specify a more complex update command contained in a file.

The update output will by default go to stdout and consists of CGE log entries. In this example, the results are redirected to a file `updates.log`:

```
$ ls -l updates.log
-rw-r--r-- 1 root 8056 May 12 11:09 updates.log
```

5. Create a checkpoint

```
$ java -cp cge-java-examples-vX.Y.Z-with-dependencies.jar \
com.cray.cge.api.examples.CheckpointExisting chkpt.sp2b.25k
```

This will checkpoint the dataset to a subdirectory in the dataset area, which in this example is named `./checkpoints/chkpt.sp2b.25k_Thu_May_12_12:00:10_CDT_2016`. Files in this directory consist of:

- `./checkpoints/chkpt.sp2b.25k_Thu_May_12_12:00:10_CDT_2016/string_table_chars.index`
- `./checkpoints/chkpt.sp2b.25k_Thu_May_12_12:00:10_CDT_2016/export_dataset.nq`
- `./checkpoints/chkpt.sp2b.25k_Thu_May_12_12:00:10_CDT_2016/string_table_chars`
- `./checkpoints/chkpt.sp2b.25k_Thu_May_12_12:00:10_CDT_2016/dbQuads`

The following will be displayed on stdout:

```
Checkpoint successful - see directory ./checkpoints
```

6. Shutdown the CGE server

```
$ java -cp cge-java-examples-vX.Y.Z-with-dependencies.jar \
com.cray.cge.api.examples.Shutdown > shutdown.log
```

The shutdown output will by default go to stdout and consists of CGE log entries. In this example, the results are redirected to a file `shutdown.log`.

```
$ ls -l shutdown.log
-rw-r--r-- 1 root 663 May 12 12:05 shutdown.log
```

15.3.4 Use Case: A Comprehensive Java Program

About this task

This procedure illustrates a use-case where Java programmers create a Java program that will execute several features of the Java API, namely:

1. Launching the CGE server
2. Running query and update commands
3. Checkpointing the dataset
4. Shutting down the CGE server.

This case utilizes the `cge-java-api-vX.Y.Z-with-dependencies.jar` file.

Procedure

1. Create an application framework:

   ```
   mvn archetype:generate \
   -DgroupId=com.cray.cge.api.examples \
   -DartifactId=my-run-cge \
   -DarchetypeArtifactId=maven-archetype-quickstart \
   -DinteractiveMode=false
   ```

   This is a standard Maven command to make a framework for developing a java application; in this case, a framework is created in the new `my-run-cge` directory area.

2. Install executable classes as a local JAR file

   ```
   mvn install:install-file \
   -Dfile=/opt/cray/cge/default/lib/java/cge-java-api-vX.Y.Z-with-dependencies.jar \
   -DgroupId=com.cray.cge.api \
   -DartifactId=cge-user-apis \
   -Dversion=1.0.0 \
   -Dpackaging=jar
   ```

   This is a standard Maven command to install an executable `.jar` file. A directory structure similar to the following should appear in the user's `.m2/repository` directory:

   ```
   ./com
   ./com/cray
   ./com/cray/cge
   ./com/cray/cge/api
   ./com/cray/cge/api/cge-user-apis
   ./com/cray/cge/api/cge-user-apis/maven-metadata-local.xml
   ./com/cray/cge/api/cge-user-apis/1.0.0
   ```

```
./com/cray/cge/api/cge-user-apis/1.0.0/cge-user-apis-1.0.0.pom
./com/cray/cge/api/cge-user-apis/1.0.0/cge-user-apis-1.0.0.jar
```

3. Switch to `my-run-cge` directory and then copy the file to `./src/main/java/com/myapp/runcge`.

```
$ cd my-run-cge
$ cp ../com/cray/cge/api/examples/ComprehensiveExample.java \
./src/main/java/com/myapp/runcge
```

4. Write the Java source code. Following is the source code for the proposed Java program used in this example. This can be copied into the `./src/main/java/com/myapp/runcge` framework area:

```
package com.cray.cge.api.examples;

// for standard java processing
import java.io.File;
import java.io.IOException;
import java.util.Collections;
import java.util.concurrent.TimeUnit;
import java.util.Date;
import org.apache.commons.lang3.StringUtils;

// for prepare cge launcher
import com.cray.cge.api.CgeConnection;
import com.cray.cge.api.CgeLauncher;
import com.cray.cge.api.builders.CgeConnectionBuilder;
import com.cray.cge.api.builders.CgeLauncherBuilder;
import com.cray.cge.api.builders.JobOptionsBuilder;
import com.cray.cge.communications.messaging.exceptions.CommunicationsException;

// for query execution
import org.apache.jena.atlas.io.IO;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.ResultSetFormatter;
import com.hp.hpl.jena.sparql.resultset.ResultsFormat;
import com.cray.cge.sparql.engine.results.ResultsMetadata;

// for update execution
import com.hp.hpl.jena.update.UpdateProcessor;

// for log4j initialization
import org.apache.log4j.Level;

/**
 * Example that demonstrates launching CGE, run query, update, checkpoint and shutdown.
 */
public class ComprehensiveExample
{
   /**
    * Default sparql commands run by this example
    */
   public static final String DEFAULT_QUERY = "SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?type } GROUP BY ?
type";
   public static final String DEFAULT_UPDATE = "INSERT DATA { <urn:s> <urn:p> <urn:o> }";

   // default runtime values the user can override with command line args.
   public static String dataset_area = "./";
   public static String output_area = "./";
   public static String checkpoint_area = "./";
   public static int NODE_COUNT=1;
   public static int IMAGE_COUNT=1;
   public static String query = DEFAULT_QUERY;
   public static String update = DEFAULT_UPDATE;
   public static String query_filename = null;
   public static String update_filename = null;
   public static File queryFile=null;
   public static File updateFile=null;
   public static int RUNTIME_TIMEOUT = 10; // minutes
   public static int STARTUP_TIMEOUT = 15; // seconds
   public static int CGE_CONNECTION_TIMEOUT = 3; // seconds
   public static int SERVER_PORT = 56789; // valid port number range: 1-65535 (1-1023 may require superuser
privileges)

   private static void showUsage()
   {
      System.out.println("\nExercises CGE by launching the server, run query, update, checkpoint, and shutdown");
      System.out.println("Usage:");
      System.out.println("   -c       CGE server-connect timeout (seconds) (default: 3)");
      System.out.println("   -d       Directory containing dataset (default: ./)");
      System.out.println("   -k       Checkpoint dataset directory (default: ./)");
      System.out.println("   -n       Number of nodes to run CGE on (default: 1)");
      System.out.println("   -i       Number of images to run CGE on (default: 1)");
      System.out.println("   -o       Outputs directory (created if does not exist) (default: ./)");
      System.out.println("   -p       CGE server port (default: " + SERVER_PORT + ") range: 1024-65535 (1-1023 as
su)");
      System.out.println("   -q       File with sparql query (default: '" + DEFAULT_QUERY + "')");
      System.out.println("   -r       Runtime timeout (minutes) (default: 10)");
      System.out.println("   -s       Startup timeout (seconds) (default: 15)");
      System.out.println("   -u       File with sparql update (default: '" + DEFAULT_UPDATE + "')");
      System.out.println("\n");
   }

   /**
    * Expects a next argument, prints an error and exists if none present
    * @param i Current Argument Index
    * @param argv Arguments
    * @param arg Current Argument for which we expect a value as the next argument
    */
   private static void expectNextArg(int i, String[] argv, String arg)
   {
      if (i >= argv.length - 1)
      {
         System.err.println("Unexpected end of arguments, expected a value to be specified after the " + arg + "
option");
         System.exit(1);
      }
   }

   /**
    * Parses Arguments
    * @param argv Arguments
    */
   private static void parseArgs(String[] argv)
   {
      for (int i = 0; i < argv.length; i++)
      {
         try
         {
            String arg = argv[i];
            if (arg.equals("-h"))
            {
               // Show Usage Summary and exit
               showUsage();
               System.exit(1);
            }
            // we have one or more name-value-pair ("NVP") args
            // (e.g., "-n 4").
            expectNextArg(i, argv, arg); // exits if "value" of the NVP is absent
            if (arg.equals("-d")) {
               dataset_area = argv[++i];
            } else if (arg.equals("-o")) {
               output_area = argv[++i];
            } else if (arg.equals("-k")) {
               checkpoint_area = argv[++i];
            } else if (arg.equals("-n")) {
               NODE_COUNT = Integer.parseInt(argv[++i]);
            } else if (arg.equals("-p")) {
               SERVER_PORT = Integer.parseInt(argv[++i]);
               if ((SERVER_PORT > 65535) || (SERVER_PORT < 1)) {
                  System.out.println("Error: arg '-p server port " + SERVER_PORT + "' value out of range");
                  System.exit(1);
               } else if (SERVER_PORT < 1024) {
                  System.out.println("Notice: arg '-p server port " + SERVER_PORT + "' may require su privileges");
               }
            } else if (arg.equals("-c")) {
               CGE_CONNECTION_TIMEOUT = Integer.parseInt(argv[++i]);
            } else if (arg.equals("-r")) {
               RUNTIME_TIMEOUT = Integer.parseInt(argv[++i]);
            } else if (arg.equals("-i")) {
               IMAGE_COUNT = Integer.parseInt(argv[++i]);
            } else if (arg.equals("-q")) {
               query_filename = argv[++i];
               queryFile = new File(query_filename);
               if (!queryFile.isFile()) {
                  System.out.println("Problem with " + query_filename + " - does not exist or not a file");
                  System.exit(1);
               }
            } else if (arg.equals("-u")) {
               update_filename = argv[++i];
               updateFile = new File(update_filename);
               if (!updateFile.isFile()) {
                  System.out.println("Problem with " + update_filename + " - does not exist or not a file");
                  System.exit(1);
               }
            } else {
               System.err.println("Illegal Option " + arg);
               showUsage();
               System.exit(1);
            }
         }
         catch (NumberFormatException numEx)
         {
            //Occurs when a numeric parameter is expected but not received
            System.err.println("Illegal value '" + argv[i] + "' encountered after option " + argv[i-1] + " when an
integer value was expected");
            System.exit(1);
         }
      }
   }


   // main entry point
   public static void main(String[] args) throws IOException, CommunicationsException, InterruptedException {

   // suppress "log4j WARN" messages
   org.apache.log4j.Logger.getRootLogger().setLevel(org.apache.log4j.Level.OFF);

   parseArgs(args);

   String DB_LOG = "cge_runtime.log";
   String LAUNCHER_LOG = "cge_launcher.log";

   // show the runtime selections
   System.out.println("CgeLauncherBuilder - start ... ");
   System.out.println("... dataset          " + dataset_area);
   System.out.println("... output area      " + output_area);
   System.out.println("... checkpoint area  " + checkpoint_area);
   System.out.println("... query file       " + query_filename);
   System.out.println("... update file      " + update_filename);
   System.out.println("... node count       " + NODE_COUNT);
   System.out.println("... image count      " + IMAGE_COUNT);
   System.out.println("... server port      " + SERVER_PORT);
   System.out.println("... run timeout      " + RUNTIME_TIMEOUT);
   System.out.println("... start timeout    " + STARTUP_TIMEOUT);
   System.out.println("... connect timeout  " + CGE_CONNECTION_TIMEOUT);

   // Prepare the launcher
   CgeLauncher launcher = new CgeLauncherBuilder()
                            .forExistingDatabase(dataset_area)
                            .usingOutputDirectory(output_area)
                            .usingDatabaseLogFile(DB_LOG)
                            .usingLauncherLogFile(LAUNCHER_LOG)
                            .onPort(SERVER_PORT)
                            .withJobOptions(new JobOptionsBuilder()
                                              .withNodes(NODE_COUNT)
                                              .withImagesPerNode(IMAGE_COUNT)
                                              .withMaximumRuntime(RUNTIME_TIMEOUT, TimeUnit.MINUTES)
                                              .build())
                            .build();

   System.out.println("CgeConnectionBuilder - start ...");

   CgeConnection cge = new CgeConnectionBuilder()
                        .usingLauncher(launcher)
                        .withConnectionTimeout(CGE_CONNECTION_TIMEOUT, TimeUnit.SECONDS)
                        .onHost("localhost")
                        .onPort(SERVER_PORT)
                        .build();

   System.out.println("CgeConnectionBuilder - done!");

   // Start CGE
   startCge(cge, STARTUP_TIMEOUT, false);
   if (cge.isRunning()) {
      System.out.println("CGE is running");
   } else {
      System.err.println("CGE failed to start");
   }


   // run query
   if (cge.isRunning()) {
      System.out.println("start query... ");
      if (queryFile != null) {
         query = IO.readWholeFileAsUTF8(queryFile.getAbsolutePath());
      }
      System.out.println("running query:\n\n" + query + "\n");


      ResultsMetadata results = cge.querySummary(query);
      if (results.wasSuccessful()) {
            System.out.println("query complete - see results in " + results.getLocation());
      } else {
            System.out.println("Error: query failed with: " + results.getError());
      }
   } else {
      System.err.println("CGE appears to not be running");
   }


   // run update
   if (cge.isRunning()) {
      System.out.println("start update... ");
      if (updateFile != null) {
         update = IO.readWholeFileAsUTF8(updateFile.getAbsolutePath());
      }
      System.out.println("running update:\n\n" + update + "\n" );
      // Updates are evaluated via the Apache Jena ARQ UpdateProcessor API
      UpdateProcessor up = cge.update(update);
      up.execute();
      System.out.println("update complete - see " + DB_LOG + " for log entries.");
   } else {
       System.err.println("CGE does not appear to be running");
   }


   // run checkpoint
   if (cge.isRunning()) {
      System.out.println("start checkpoint... ");
      // Checkpoint the database currently in use by cge-server.
      Date curr_date = new Date();
      String clean_date = curr_date.toString();
      File cpDir = new File(checkpoint_area, "checkpoint" + File.separator + dataset_area.replace('/','_') + "_" +
clean_date.replace(' ', '_'));
      cge.checkpoint(cpDir, true);
      System.out.println("Checkpoint successful - see directory " + checkpoint_area + "/checkpoint");
   } else {
      System.err.println("CGE failed to start");
   }


   // Shutdown
   if (cge.isRunning()) {
      System.out.println("start shutdown... ");
      cge.stop();
      cge.getProcess().waitFor();
      System.out.println("...shutdown complete");
   }


   System.out.println( "exiting...");
   System.exit(0);
   }


   /**
    * Starts the CGE instance represented by the given connection
    *
    * @param cge
    *            CGE connection
    * @param maxWaitSeconds
    *            Maximum number of seconds to wait for start up
    * @param returnOnInterrupt
    *            Whether to return if interrupted while waiting
    * @throws IOException
    *             Thrown if there is a problem starting CGE
    */
   public static void startCge(CgeConnection cge, int maxWaitSeconds, boolean returnOnInterrupt) throws
IOException {

      System.out.println("Starting CGE...");
      cge.start();

      long startTime = System.currentTimeMillis();
      while (TimeUnit.MILLISECONDS.toSeconds(System.currentTimeMillis() - startTime) < maxWaitSeconds)
      {
         try
         {
            Thread.sleep(1000);
         }
         catch (InterruptedException e)
         {
            // Ignore or return as appropriate
            if (returnOnInterrupt)
            {
               System.out.println(String.format("Interrupted while waiting for CGE to ready (%d seconds elapsed)",
                                  TimeUnit.MILLISECONDS.toSeconds(System.currentTimeMillis() - startTime)));
               return;
            }
         }
         if (cge.isRunning())
         {
            System.out.println(String.format("CGE ready in %d seconds",
                               TimeUnit.MILLISECONDS.toSeconds(System.currentTimeMillis() - startTime)));
            return;
         }
         System.out.println(String.format("CGE not yet ready (%d seconds elapsed)",
                            TimeUnit.MILLISECONDS.toSeconds(System.currentTimeMillis() - startTime)));
      }
   }
}
```

5. Use the following `pom.xml` file, which is developed for building application code:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.cray.cge.api.examples</groupId>
  <artifactId>my-run-cge</artifactId>
  <packaging>jar</packaging>
  <version>1.0.0</version>
  <name>my-run-cge</name>
  <url>http://maven.apache.org</url>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>2.6</version>
          <executions>
            <execution>
               <goals>
                  <goal>attached</goal>
               </goals>
               <phase>package</phase>
               <configuration>
                 <descriptorRefs>
                   <descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
                <archive>
                  <manifest>
                    <mainClass>com.cray.cge.api.examples.ComprehensiveExample</mainClass>
                  </manifest>
                </archive>
              </configuration>
           </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.cray.cge.api.spark</groupId>
        <artifactId>cge-user-apis</artifactId>
        <version>1.0.0</version>
    </dependency>
  </dependencies>
</project>
```

6. Build

```
$ mvn clean package
```

Users can first delete all files in their `~/.m2` directory, as Maven will download what it needs to build the package into this area.

7. Execute the `help` command. This will show the help menu for the application, defining the optional runtime arguments (note that all have default values):

```
system:~/comprehensive_test/my-run-cge> java -jar target/my-run-cge-1.0.0-jar-with-dependencies.jar -h
 Exercises CGE by launching the server, run query, update, checkpoint, and shutdown
Usage:
   -c       CGE server-connect timeout (seconds) (default: 3)
   -d       Directory containing dataset (default: ./)
```

```
    -k       Checkpoint dataset directory (default: ./)
    -n       Number of nodes to run CGE on (default: 1)
    -i       Number of images to run CGE on (default: 1)
    -o       Outputs directory (created if does not exist) (default: ./)
    -p       CGE server port (default: 56789) range: 1024-65535 (1-1023 as su)
    -q       File with sparql query (default: 'SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?type } GROUP
 BY ?type')
    -r       Runtime timeout (minutes) (default: 10)
    -s       Startup timeout (seconds) (default: 15)
    -u       File with sparql update (default: 'INSERT DATA { <urn:s> <urn:p> <urn:o> }')
```

8. Execute the code.

The following is a sample execution command and resulting output:

```
system:~/comprehensive_test/my-run-cge> java -jar target/my-run-cge-1.0.0-jar-with-dependencies.jar -d \
/lus/scratch/ripple/mkdb/sp2b/25k -p 12345 -r 2 -i 4 -n 3 -k /lus/scratch/temp -q sp2b_query_9.txt
CgeLauncherBuilder - start ...
... dataset          /lus/scratch/ripple/mkdb/sp2b/25k
... output area      ./
... checkpoint area  /lus/scratch/temp
... query file       sp2b_query_9.txt
... update file      null
... node count       3
... image count      4
... server port      12345
... run timeout      2
... start timeout    15
... connect timeout  3
CgeConnectionBuilder - start ...
CgeConnectionBuilder - done!
Starting CGE...
CGE not yet ready (1 seconds elapsed)
CGE not yet ready (2 seconds elapsed)
CGE not yet ready (3 seconds elapsed)
CGE ready in 7 seconds
CGE is running
start query...
running query:

PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?predicate
WHERE {
  {
    ?person rdf:type foaf:Person .
    ?subject ?predicate ?person
  } UNION {
    ?person rdf:type foaf:Person .
    ?person ?predicate ?object
  }
}

query complete - see results in file: /lus/scratch/comprehensive_test/my-run-cge/queryResults.<YEAR-DAY-TIME>.tsv
start update...
running update:

INSERT DATA { <urn:s> <urn:p> <urn:o> }

update complete - see cge_runtime.log for log entries.
start checkpoint...
Checkpoint successful - see directory /lus/scratch/temp/checkpoint
start shutdown...
...shutdown complete
exiting...
```

The dataset being referenced in this example is `sp2b-25k`, and resides in the `/lus/scratch/ripple/mkdb/sp2b/25k` directory. Three nodes and four images were specified for the CGE server to execute on. Two minutes were allowed for the execution, 15 seconds for the startup, and three seconds to connect to the server, which should have been started at port 12345. The CGE server began executing within seven seconds of the initial command. The query being run was in the local file `sp2b_query_9.txt`, the content of which is shown following the "running query" banner. The output area for the query results was the local directory `./`. An update command file is not specified and so the default was used. The checkpointed dataset went into the `/lus/scratch/temp/checkpoint` directory under a sub-directory that in this case was named `_lus_scratch_ripple_mkdb_sp2b_25k`*DAY_MONTH_DATE_TIME_ZONE_YEAR*. The shutdown would have removed all processes started by the Java execution, as could be verified by running `ps aux |grep $USER` from a command line after the "`exiting...`" message appears.

15.3.5 Limitations of CGE Java API

Following are some limitations that should be kept under consideration when using the CGE Java API.

Network access requirements

Connecting to the database requires network access from the machine running the API to the node where the database is running. If there is no such access, no operations can be carried out.

Database Launch

Launching a database relies upon being able to use the `cge-launch` command, this imposes two key limitations:

1. The command must be visible on the user's `$PATH` or the `$PATH` of the execution environment in order to launch a database, where `$PATH` is an environment variable.
2. Launching a database can only be carried out on the system i.e., if a user is running code that uses the API on a remote system, the user will not be able to launch databases.
3. Launching a database cannot be done from inside a pre-existing resource allocation since `cge-launch` expects to obtain the resource allocation itself. Launching a database should only be done from code running on a login node

Accessing query results

When executing queries, the database writes the results to the configured file system. In order to retrieve those results from the API, access is required to the same file system and sufficient privileges are required to read those result files.

If queries are being executed on a machine without access to the configured file system, the user will only be able to access meta data about the results, not the results themselves.

Log capture

The log capture functionality of the API relies upon access to the database log file. If that file is not known or not accessible (for example if the user is running on a remote machine), it will not be possible to retrieve log entries.

Note that the API will inject a unique identifier into the logs for each operation carried out via the API. Therefore it is possible to extract the log entries at a later date because the API will still be able to indicate the unique identifier used, which can be stored for later reference.

15.4 CGE Python API

This feature is currently supported on Urika-GX only.

The CGE Python API allows users to run CGE operations from their python applications on login nodes. The Python API can start the CGE server, run a query, update, checkpoint, and shutdown. Python users will indirectly utilize the CGE Java API in a Java Virtual Machine (JVM) - the 'py4j' component provides the gateway to that JVM. Essentially, a user's python application will function as a front end UI - users can be as spare or elaborate as desired in their python applications for starting the CGE server, managing their queries, updates and checkpoints, and displaying query results.

CGE Python API Components

CGE Python API components and their locations on the Urika-GX system are listed below:

- `/usr/share/py4j/py4j0.10.3.jar` - This is version 0.10.3 of the py4j package which implements a python to Java bridge.
- `/usr/lib/python2.7/site-packages/py4j/` - This is the py4j python code that executes under python version 2.7
- `/usr/lib/python3.4/site-packages/py4j/` - This is the py4j python code that executes under python version 3.4
- `/opt/cray/cge/default/lib/java/cge-java-api-vX.Y.Z-with-dependencies.jar` - This is the CGE Java API.
- `/opt/cray/cge/default/lib/python/cge_python_api-vX.Y.Z.py` - This is an API example that can run as a python application or in the python interpreter.

The `py4j` Package

The 'py4j' component of the Python API is an open-source package that enables python programs running in a python interpreter to dynamically access Java objects running in a Java Virtual Machine (JVM). The Python API utilizes this package to access the CGE Java API to build Job Options, launch the CGE server, setup runtime locations for CGE logs and query output, execute queries and updates, etc. It consists of a Java .jar file, and `.py` files that can execute under the two python versions shown.

Although the `py4j` package is located at `/usr/share/py4j/py4j0.10.3.jar` on the system, the location of `py4j` used in the examples of this guide can be tailored according to site requirements. Moreover, users can specify any version of the `py4j` JAR file that they wish to utilize. The sample code provided in this guide implies this by showing an explicit path to the `py4j` JAR file, the implication being that users can use a different `py4j` JAR file at a different location at their discretion.

Detailed documentation of this package can be found at *https://www.py4j.org*

15.4.1 Use Case: A Comprehensive Python Program

This is the `cge_python_api-vX.Y.Z.py` component of the Python API that shows how to interact with CGE. This sample will start CGE, run a query, an update, checkpoint, and shutdown.

```
# Copyright 2016 Cray Inc. All Rights Reserved.
 #
 # (c) Cray Inc.  All Rights Reserved.  Unpublished Proprietary
 # Information.  This unpublished work is protected by trade secret,
 # copyright and other laws.  Except as permitted by contract or
 # express written permission of Cray Inc., no part of this work or
 # its content may be used, reproduced or disclosed in any form.

"""CGE Python API allows users to run CGE from their python applications.
Python users will transparently utilize the CGE Java API in a JVM (Java
Virtual Machine) - the 'py4j' package referenced here provides the gateway
to that JVM. Essentially, a user's python application will function as a
front-end UI - users can be as spare or elaborate as desired in their python
applications for starting the cge-server, managing their queries updates
```

```
and checkpoints, and interpreting and displaying query results. This
example shows how to start cge-server, run a query, update, checkpoint, and
shutdown. It is meant to form the basis for more elaborate user python apps.
"""
__version__ = '0.1'
__revision__ = '$Revision:$'
__all__ = ['Server', '__version__', '__revision__']

# bring in standard objects
import time
import os


#--- bring in the py4j JVM gateway objects
from py4j.java_gateway import JavaGateway
from py4j.java_gateway import java_import

#--- start the Java GatewayServer in a JVM (explicit paths to the jar files)
gateway = JavaGateway.launch_gateway(
    jarpath='/share/py4j/py4j0.10.3.jar',
    classpath='/opt/cray/cge/default/lib/java/cge-java-api-v1.1.0-with-
dependencies.jar')

#--- bring in some commonly used items
java_import(gateway.jvm,'com.cray.cge.api.builders.*')
my_timeunit = gateway.jvm.java.util.concurrent.TimeUnit

#--- these can be modified as desired for different port,
#--- node count and images per node.
MY_CGE_SERVER_PORT = 23239
MY_NODE_COUNT = 2
MY_IMAGE_COUNT = 6

#--- build the JobOptions
my_cge_joboptions_builder =
gateway.jvm.com.cray.cge.api.builders.JobOptionsBuilder()
my_cge_joboptions_builder.withNodes(MY_NODE_COUNT)
my_cge_joboptions_builder.withImagesPerNode(MY_IMAGE_COUNT)
#--- runtime timeout can be changed as desired
RUNTIME_TIMEOUT_MINUTES = 60
my_cge_joboptions_builder.withMaximumRuntime(RUNTIME_TIMEOUT_MINUTES,
my_timeunit.MINUTES)

#--- get the job options
my_cge_joboptions = my_cge_joboptions_builder.build()

#--- read back and show the options
readback_nodes = my_cge_joboptions.getNodes()
print "read back: nodes=",readback_nodes
readback_imagesPerNode = my_cge_joboptions.getImagesPerNode()
print "read back: images per node=",readback_imagesPerNode
readback_totalImages = my_cge_joboptions.getTotalImages()
print "read back: total images=",readback_totalImages

#--- build the launcher-builder
my_cge_launcher_builder =
gateway.jvm.com.cray.cge.api.builders.CgeLauncherBuilder()
#--- specify dataset location (sample shown)
my_cge_launcher_builder.forExistingDatabase("/mnt/lustre/ripple/mkdb/sp2b/25k")
#--- place query output files into current working dir.
cwd = os.getcwd()
```

```
my_cge_launcher_builder.usingOutputDirectory(cwd)
#--- the cge runtime and launcher log will go into the current working dir.
my_cge_launcher_builder.usingDatabaseLogFile("cge_runtime.log")
my_cge_launcher_builder.usingLauncherLogFile("cge_launcher.log")
my_cge_launcher_builder.onPort(MY_CGE_SERVER_PORT)
my_cge_launcher_builder.withJobOptions(my_cge_joboptions)

#--- build the launcher-builder and get the launcher
my_cge_launcher = my_cge_launcher_builder.build()

#--- build the connection-builder
my_cge_conn_builder = gateway.jvm.com.cray.cge.api.builders.CgeConnectionBuilder()
#--- allow 15 second startup timeout (make larger if desired)
my_cge_conn_builder.withConnectionTimeout(15, my_timeunit.SECONDS)
my_cge_conn_builder.onHost("localhost")
my_cge_conn_builder.onPort(MY_CGE_SERVER_PORT)
my_cge_conn_builder.nonInteractive()
my_cge_conn_builder.trustHostKeys()

#--- make the connection
my_cge_conn_builder.usingLauncher(my_cge_launcher)
my_conn = my_cge_conn_builder.build()

#--- read back and show the options
readback_port = my_conn.getPort()
print "read back port=",readback_port
readback_host = my_conn.getHost()
print "read back host=",readback_host

#--- time stamp the start time
CGE_STARTUP_TIMEOUT_SECONDS = 1000
ONE_SECOND = 1
start = time.time()

#--- start cge
my_conn.start()

#--- poll 'isRunning()' for the signal that cge has started
#--- (sleep a second between polls to minimize processing)
while True:
    time.sleep(ONE_SECOND)
    delta = time.time() - start
    if delta >= CGE_STARTUP_TIMEOUT_SECONDS:
        print "CGE did not start"
        #--- kill the Java JVM
        gateway.shutdown()
        exit()
    if my_conn.isRunning() == True:
        print "CGE started ok!"
        break

#--- look at cge status another way
java_import(gateway.jvm,'com.cray.cge.api.status.*')
my_CgeStatus = my_conn.status()
runtime_status = my_CgeStatus.toString()
print "runtime status=",runtime_status

#--- a simple query
DEFAULT_QUERY = "SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?type } GROUP
BY ?type"
print DEFAULT_QUERY
```

```
#--- run the query against the dataset specified above
my_query_results = my_conn.querySummary(DEFAULT_QUERY)

#--- wait for query completion
my_query_results.wasSuccessful()

#--- get name of results file
my_query_results.getLocation()

#--- a simple update command
DEFAULT_UPDATE = "INSERT DATA { <urn:s> <urn:p> <urn:o> }"
print DEFAULT_UPDATE

#--- run the update
my_update_cmd = my_conn.update(DEFAULT_UPDATE)
my_update_cmd.execute()

#--- run checkpoint, place in current working dir.
my_conn.checkpoint(cwd, False)

#--- shutdown cge
my_conn.stop()

#--- wait for cge to shutdown
my_conn.getProcess().waitFor()

#--- kill the Java JVM
gateway.shutdown()
```

Although the code can be run as-is, or with a more complex query and update defined in place of the simple query and update shown, the program is meant to be a guide to more elaborate code development specific to the user's requirements. For example, at the point where the "`DEFAULT_QUERY`" is defined and printed, users could develop a more sophisticated query management technique for acquiring complex queries from files and looping through their execution. Similarly for updates and checkpoints. The selection of `MY_NODE_COUNT` and `MY_IMAGE_COUNT` could incorporate a UI for more interactive selection of those values. And so forth for other sections of the code.

In general, the use of gateway.jvm.com.cray.cge.api and the functions referenced must be invoked in the order shown and with equivalent arguments. In between those function invocations, users can be as elaborate or spare as their applications require. When running under Python-3, the arguments to "`print`" statements need to be placed in parenthesis. For example: print "`read back: nodes=`", `readback_nodes`, should be changed to print ("`read back: nodes=`", `readback_nodes`)

15.4.2 Run the CGE Python API as a Python Application

To run this code as a python application on a login node, enter the command `python cge_python_api-v`*`X.Y.Z`*`.py`, replacing *`X.Y.Z`* with the corresponding version number. The current version number of the Python API is 1.0.0

Here is an example of output that will appear:

```
[userid@nid00030~]$ python cge_python_api-v1.0.0.py
read back: nodes= 2
read back: images per node= 6
read back: total images= 12
read back port= 23239
read back host= localhost
CGE started ok!
runtime status= Process: Running - CGE: Running
SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?type } GROUP BY ?type
INSERT DATA { <urn:s> <urn:p> <urn:o> }
[userid@nid00030 ~]$
```

- The `read back` values show the user's selections
- The `total images` is computed by CGE and shown. The print, "`CGE started ok!`" indicates that the CGE server started successfully on the specified dataset, within the timeout argument values, with the given node and image count.
- The runtime status is shown as `Running`.
- The program's sample `query` command is shown in the print of the `SELECT` statement.
- The program's sample `update` command is shown in the print of the `INSERT` statement.

Example file outputs from the program:

```
[userid@nid00030 ~]$ ls -lt
total 2256
-rw-r--r-- 1 userid hw    1473 Sep 27 15:54 cge_launcher.log
-rw-r--r-- 1 userid hw  129254 Sep 27 15:54 cge_runtime.log
-rw-r--r-- 1 userid hw    3675 Sep 27 15:54 rules.txt
-rw-r--r-- 1 userid hw 1413120 Sep 27 15:54 string_table_chars
-rw-r--r-- 1 userid hw    8192 Sep 27 15:54 string_table_chars.index
-rw-r--r-- 1 userid hw  671208 Sep 27 15:54 dbQuads
-rw-r--r-- 1 userid hw     769 Sep 27 15:54 queryResults.
2016-09-27T20.54.59Z000.8006.tsv
```

- The `*.log` files are produced by CGE.
- The `rules.txt`, `string_table*`, and `dbQuads` file are the files of the checkpointed example dataset.
- The `queryResults*.tsv` file is the output of the `SELECT` query.

The following are user processes active when running the Python program:

```
[userid@nid00030 ~]$ top -u $USER
top - 16:09:17 up 47 days,  1:22, 39 users,  load average: 0.08, 0.09, 0.38
Tasks: 789 total,   2 running, 787 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 52914899+total, 33347744+free,  9836832 used, 18583470+buff/cache
KiB Swap:        0 total,        0 free,        0 used. 49864905+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
57794 userid   20   0  273696   8148   3508 S   0.3  0.0   0:00.02 cge-launch
57796 userid   20   0  249036  37280   4032 S   0.0  0.0   0:00.65 mrun
57732 userid   20   0  344940  11368   3580 S   0.0  0.0   0:00.05 python
57733 userid   20   0 35.916g 126492  15576 S   0.0  0.0   0:01.98 java
```

The Python process launched the Java JVM process with the `gateway = JavaGateway.launch_gateway(..)` command. The Java JVM process is exited with the `gateway.shutdown()` command.

The `cge-launch` and `mrun` processes are the runtime signature of CGE, which was launched with the `my_conn.start()` command. These processes are exited with the `my_conn.stop()` command.

15.4.3 Run a Python API from the Python Interpreter

The Python API can be run from the python interpreter by copy-paste of the program into the interpreter. Processes started and outputs produced are the same as shown above. For example, here is a sample run of the code from the python interpreter, with the interpreter's responses shown:

```
[userid@nid00030 ~]$ python
Python 2.7.5 (default, Nov 20 2015, 02:00:19)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>> # bring in standard objects
... import time
>>> import os
>>>
>>>
>>> #--- bring in the py4j JVM gateway objects
... from py4j.java_gateway import JavaGateway
>>> from py4j.java_gateway import java_import
>>>
>>>
>>> #--- start the Java GatewayServer in a JVM (explicit paths to the jar files)
... gateway = JavaGateway.launch_gateway(
...     jarpath='/usr/share/py4j/py4j0.10.3.jar',
...     classpath='/opt/cray/cge/default/lib/java/cge-java-api-v1.1.0-with-
dependencies.jar')
>>>
>>>
>>> #--- bring in some commonly used items
... java_import(gateway.jvm,'com.cray.cge.api.builders.*')
>>> my_timeunit = gateway.jvm.java.util.concurrent.TimeUnit
>>>
>>>
>>> #--- these can be modified as desired for different port,
... #--- node count and images per node.
... MY_CGE_SERVER_PORT = 23239
>>> MY_NODE_COUNT = 2
>>> MY_IMAGE_COUNT = 6
>>>
>>>
>>> #--- build the JobOptions
... my_cge_joboptions_builder =
gateway.jvm.com.cray.cge.api.builders.JobOptionsBuilder()
>>>
>>> my_cge_joboptions_builder.withNodes(MY_NODE_COUNT)
JavaObject id=o1
>>> my_cge_joboptions_builder.withImagesPerNode(MY_IMAGE_COUNT)
JavaObject id=o2
>>> #--- runtime timeout can be changed as desired
... RUNTIME_TIMEOUT_MINUTES = 60
>>> my_cge_joboptions_builder.withMaximumRuntime(RUNTIME_TIMEOUT_MINUTES,
my_timeunit.MINUTES)
JavaObject id=o4
>>>
>>> #--- get the job options
... my_cge_joboptions = my_cge_joboptions_builder.build()
>>>
>>> #--- read back and show the options
... readback_nodes = my_cge_joboptions.getNodes()
```

```
>>> print "read back: nodes=",readback_nodes
read back: nodes= 2
>>> readback_imagesPerNode = my_cge_joboptions.getImagesPerNode()
>>> print "read back: images per node=",readback_imagesPerNode
read back: images per node= 6
>>> readback_totalImages = my_cge_joboptions.getTotalImages()
>>> print "read back: total images=",readback_totalImages
read back: total images= 12
>>>
>>> #--- build the launcher-builder
... my_cge_launcher_builder =
gateway.jvm.com.cray.cge.api.builders.CgeLauncherBuilder()
>>> #--- specify dataset location (sample shown)
... my_cge_launcher_builder.forExistingDatabase("/mnt/lustre/ripple/mkdb/sp2b/
25k")
#--- place query output files into current working dir.
cwd = os.getcwd()
my_cge_launcher_builder.usingOutputDirectory(cwd)
#--- the cge runtime and launcher log will go into the current working dir.
my_cge_launcher_builder.usingDatabaseLogFile("cge_runtime.log")
my_cge_launcher_builder.usingLauncherLogFile("cge_launcher.log")
my_cge_launcher_builder.onPort(MY_CGE_SERVER_PORT)
my_cge_launcher_builder.withJobOptions(my_cge_joboptions)
JavaObject id=o7

>>> #--- place query output files into current working dir.
... cwd = os.getcwd()
>>> my_cge_launcher_builder.usingOutputDirectory(cwd)
JavaObject id=o8
>>> #--- the cge runtime and launcher log will go into the current working dir.
... my_cge_launcher_builder.usingDatabaseLogFile("cge_runtime.log")
JavaObject id=o9
>>> my_cge_launcher_builder.usingLauncherLogFile("cge_launcher.log")
JavaObject id=o10
>>> my_cge_launcher_builder.onPort(MY_CGE_SERVER_PORT)
JavaObject id=o11
>>> my_cge_launcher_builder.withJobOptions(my_cge_joboptions)
JavaObject id=o12
>>>
>>>
>>> #--- build the launcher-builder and get the launcher
... my_cge_launcher = my_cge_launcher_builder.build()
>>>
>>>
>>> #--- build the connection-builder
... my_cge_conn_builder =
gateway.jvm.com.cray.cge.api.builders.CgeConnectionBuilder()
>>> #--- allow 15 second startup timeout (make larger if desired)
... my_cge_conn_builder.withConnectionTimeout(15, my_timeunit.SECONDS)
JavaObject id=o16
>>> my_cge_conn_builder.onHost("localhost")
JavaObject id=o17
>>> my_cge_conn_builder.onPort(MY_CGE_SERVER_PORT)
my_cge_conn_builder.nonInteractive()
JavaObject id=o18
>>> my_cge_conn_builder.nonInteractive()
my_cge_conn_builder.trustHostKeys()
JavaObject id=o19
>>> my_cge_conn_builder.trustHostKeys()
JavaObject id=o20
>>>
```

```
>>>
>>> #--- make the connection
... my_cge_conn_builder.usingLauncher(my_cge_launcher)
JavaObject id=o21
>>> my_conn = my_cge_conn_builder.build()
>>>
>>> #--- read back and show the options
... readback_port = my_conn.getPort()
>>> print "read back port=",readback_port
read back port= 23239
>>> readback_host = my_conn.getHost()
>>> print "read back host=",readback_host
read back host= localhost
>>>
>>> #--- time stamp the start time
... CGE_STARTUP_TIMEOUT_SECONDS = 1000
>>> ONE_SECOND = 1
>>> start = time.time()
>>>
>>> #--- start cge
... my_conn.start()
>>>
>>>
>>> #--- poll 'isRunning()' for the signal that cge has started
... #--- (sleep a second between polls to minimize processing)
... while True:
...     time.sleep(ONE_SECOND)
...     delta = time.time() - start
...     if delta >= CGE_STARTUP_TIMEOUT_SECONDS:
...         print "CGE did not start"
...         #--- kill the Java JVM
...         gateway.shutdown()
...         exit()
...     if my_conn.isRunning() == True:
...         print "CGE started ok!"
...         break
...
CGE started ok!
>>>
>>>
>>> #--- look at cge status another way
... java_import(gateway.jvm,'com.cray.cge.api.status.*')
>>> my_CgeStatus = my_conn.status()
>>> runtime_status = my_CgeStatus.toString()
>>> print "runtime status=",runtime_status
runtime status= Process: Running - CGE: Running
>>>
>>> #--- a simple query
... DEFAULT_QUERY = "SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?type }
GROUP BY ?type"
>>> print DEFAULT_QUERY
SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?type } GROUP BY ?type
>>>
>>>
>>>
>>> #--- run the query against the dataset specified above
... my_query_results = my_conn.querySummary(DEFAULT_QUERY)
>>>
>>> #--- wait for query completion
... my_query_results.wasSuccessful()
True
```

```
>>>
>>> #--- get name of results file
... my_query_results.getLocation()
u'file:///home/users/userid/queryResults.2016-09-27T21.59.35Z000.31323.tsv'
>>>
>>>
>>> #--- a simple update command
... DEFAULT_UPDATE = "INSERT DATA { <urn:s> <urn:p> <urn:o> }"
>>> print DEFAULT_UPDATE
INSERT DATA { <urn:s> <urn:p> <urn:o> }
>>>
>>> #--- run the update
... my_update_cmd = my_conn.update(DEFAULT_UPDATE)
>>> my_update_cmd.execute()
>>>
>>>
>>> #--- run checkpoint, place in current working dir.
... my_conn.checkpoint(cwd, False)
>>>
>>>
>>> #--- shutdown cge
... my_conn.stop()
>>>
>>> #--- wait for cge to shutdown
... my_conn.getProcess().waitFor()
0
>>>
>>>
>>> #--- kill the Java JVM
... gateway.shutdown()
>>>
>>>
```

15.5 CGE Spark API

CGE works with RDF and generates its results files in the form of an array of tab-separated values, that are either identifiers or literals. This allows Spark programmers to convert CGE results files in tab-separated-values format ("`.tsv`" files) into Spark datasets. Spark programmers can write a schema in the corresponding Spark language (Scala, Java, R, Python) that represents the columns present in the TSV file, and then invoke a function in the CGE Java API with the schema and path to the TSV file as arguments. The function returns a Spark dataset conforming to the schema. From there, the Spark programmer can perform Spark queries and transformations on the dataset in Spark context.

Alternatively, the Spark programmer may elect to convert the TSV file to a Spark dataset where all the column entries of the TSV file will be interpreted as strings. This produces a Spark dataset, where all the columns exactly reproduce the content of the TSV file as string columns. The CGE Spark API feature allows Spark programmers to save the contents of their Spark DataFrame in RDF format. After reading in a TSV file, a Spark programmer can process the DataFrame using Spark facilities and then save the data in RDF format. The saved data can then be read into CGE for further processing.

Spark Execution Command

The CGE Spark API is a set of Java functions built into the CGE Java API. To use the API, Spark users need to launch their Spark sessions referencing the CGE Java API jar file, where the "v*X*.*Y*.*Z*" is to be replaced by the current version number of the CGE Java API.

Examples:

In the following, `X.Y.Z` is used as an example for the CGE Java API version and should be replaced with the actual version number. Similarly, `path` should be replaced with the actual paths when using these examples.

Spark Scala

```
$ spark-shell --jars path/cge-java-api-vX.Y.Z-with-dependencies.jar --conf
"spark.debug.maxToStringFields=38"
```

Spark Python (for Urika-GX only)

```
$ pyspark --jars path/cge-java-api-vX.Y.Z-with-dependencies.jar
```

Spark R

```
$ sparkR --jars path/cge-java-api-vX.Y.Z-with-dependencies.jar
```

Spark Java

```
$ spark-submit  --class "user.class.path"  user_executable.jar  optional_program_arguments
```

In the preceding example, `user_executable.jar` must be built with a dependency on the CGE Java API, for example:

```
<dependencies>
  <dependency>
    <groupId>com.cray.cge.api.spark</groupId>
    <artifactId>cge-user-apis</artifactId>
    <version>X.Y.Z</version>
  </dependency>
</dependencies>
```

15.5.1 Convert TSV Files to Spark Datasets

The CGE Spark API enables converting TSV files to Spark datasets.

Conversion of TSV Files to Spark Datasets with All Strings Columns

Spark programmers can convert a TSV file to a Spark dataset by invoking the `getSparkDataset()` API routine. Each column of the resultant Spark dataset will be a 1-to-1 mapping of columns from the TSV file, where all the entries in the columns are strings taken verbatim from the TSV file.

The syntax of the `getSparkDataset()` function is:

```
getSparkDataset(String tsv-file-path, Boolean showProgress)
```

Where:

- *`tsv-file-path`* is the path to the TSV file to convert:
  - Local filespace: ("`file:///`*`path/filename`*`.tsv`")
  - HDFS filespace: ("`/`*`path/filename`*`.tsv`")
- *`showProgress`*: Setting this to `true=enable` will display statements in the API to show progress messages during the conversion, whereas setting it to `false=silent` will not display and messages during the conversion.

Examples

Spark Scala

```
val df = com.cray.cge.api.spark.SparkCgeApi.getSparkDataset("/path/file.tsv", true)
```

Spark Python (for Urika-GX only)

```
from py4j.java_gateway import java_import
from pyspark import SparkContext
jvm = sc._gateway.jvm
java_import(jvm, "com.cray.cge.api.spark.SparkCgeApi.*")
.
.
.
show_progress = True
local_tsv_file = "file:///path/file.tsv"
ds =
jvm.com.cray.cge.api.spark.SparkCgeApi.getSparkDataset(local_tsv_file
,show_progress)
```

Spark R

```
show_progress = True
local_tsv_file = "file:///path/file.tsv"
ds_1 =
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","getSparkDatas
et",local_tsv_file, TRUE)
```

Spark Java:

```
import org.apache.spark.sql.*;
import com.cray.cge.api.spark.*;
.
.
.
// make an instance of the SparkCgeApi.
SparkCgeApi the_api = new SparkCgeApi();
show_progress = true;
Dataset<Row> df = the_api.getSparkDataset("/path/file.tsv",
show_progress);
```

Conversion of TSV Files to Spark Datasets with Parsed Columns

Spark programmers can convert a TSV file to Spark dataset by Invoking the `getSparkDatasetUsingSchema()` API routine. Each column of the resultant dataset will be a 1-to-1 mapping of columns from the TSV file, where all the entries in the columns are parsed according to a schema written by the Spark programmer.

The syntax of the `getSparkDatasetUsingSchema()` function is:

```
getSparkDatasetUsingSchema(String tsv-file-path, StructType schema, Boolean retainOriginalColumns)
```

Where:

- *`tsv-file-path`* is the path to the TSV file to convert:
  - Local filespace: (`file:///`*`path/filename`*`.tsv`)
  - HDFS filespace: (`/`*`path/filename`*`.tsv`
- *`schema`* defines the column entries to expect in the TSV-to-dataset conversion
- `retainOriginalColumns` - Setting the value of this option to `true` copies the columns of the TSV file into the resultant dataset in their original string format, whereas setting it to `false` discards the original columns.

The `retainOriginalColumns` argument, when set `true`, will cause all the original columns from the TSV file to be retained in the resultant dataset. This for use, for example, in the event that a TSV column contains mixed literal types, which will produce null column entries for types that do not match the type expected by the schema. Using Spark `select` statements on the resultant dataset, the Spark programmer can examine the original TSV data next to the translated data to see the effect of their schema on the resultant dataset.

15.5.2 Scheme Determination

The Spark programmer first needs to examine the CGE results TSV file that needs to be transformed in order to write the schema that will direct the TSV-to-Spark-dataset transformation. The Spark programmer also needs to know the number of columns the TSV file contains, and what the columns might contain, i.e., either URL strings, literal types or mixed combinations.

Number of Columns in the TSV file

All CGE results TSV files contain a heading in the first line of the file. For example:

```
?gs    ?pub    ?dept    ?prof1    ?crs
```

This heading line identifies five columns in the TSV file, column names shown. The heading line will always consist of tab-separated values, the number of which correspond with the number of columns in the file, but will not indicate the type of data in each column.

URL Lines

These type of column entries can contain any text. For example:

- `<http://www.Department4.University4614.edu>`
- `"GraduateStudent99@Department12.University0.edu"`
- `"AssociateProfessor3"`

Literal Types

Column entries of literal types always contain the following text:

`http://www.w3.org.2001/XMLSchema#`

Here are some example literal types the Spark programmer can expect to find in tsv file columns:

- `"11"^^<http://www.w3.org/2001/XMLSchema#integer>`
- `"9.1"^^<http://www.w3.org/2001/XMLSchema#float>`
- `"-3E2"^^<http://www.w3.org/2001/XMLSchema#double>`
- `"12345678"^^<http://www.w3.org/2001/XMLSchema#long>`
- `"cafe"^^<http://www.w3.org/2001/XMLSchema#hexBinary>`
- `"-6"^^<http://www.w3.org/2001/XMLSchema#negativeInteger>`
- `"2006-08-27T09:00:00+03:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>`
- `"now is"^^<http://www.w3.org/2001/XMLSchema#string>`

A full discussion of literal types is beyond the scope of this document. For more information, see *https://www.w3.org/TR/xmlschema-2/*. The important aspect, however, is that when literal types are present in the CGE results TSV file, the Spark programmer can write schema's identifying the literal type they wish to parse to its base type in each resultant dataset column. For example, when a column contains a "float" literal type such as `"9.1"^^<http://www.w3.org/2001/XMLSchema#float>`, the quoted string containing the float value ("9.1") will, under direction by the schema, be parsed into a column of float values. The API will attempt to parse ALL entries of that particular column as float literal types. A discussion of what the API will do with a column of mixed, or inconsistent literal types, follows.

Mixed Types

This refers to columns of mixed URL and literal types, as described above. When a column of a tsv file consists of mixed types, the Spark programmer must simply choose a single literal type to declare in the schema, or declare the column a `StringType`. The former will result in the TSV column being parsed according to the data type chosen, where all other non-conforming types in the column will result in null column entries in the resultant dataset column. In the latter case, ALL literal-type entries in the column will be parsed to strings, with non-literal types retaining their string content verbatim. The best case for specifying that literals should be converted to a Spark literal type is when the user knows that all or most of the XML Schema literals in a column are of the same type.

Here is an example of a mixed-type column parsed as "StringType":

```
+--------------------------------------------------------+-----------------------------------------+
|?mixedtypes                                             |mixedtypes                               |
+--------------------------------------------------------+-----------------------------------------+
|<http://www.Department10.University0.edu>               |<http://www.Department10.University0.edu>|
|"5678"^^<http://www.w3.org/2001/XMLSchema#integer>      |5678                                     |
|"5.678"^^<http://www.w3.org/2001/XMLSchema#float>       |5.678                                    |
|"tobeornot."^^<http://www.w3.org/2001/XMLSchema#string> |tobeornot.                               |
|"2001-99-99"^^<http://www.w3.org/2001/XMLSchema#date>   |2001-99-99                               |
|"somelong"^^<http://www.w3.org/2001/XMLSchema#long>     |somelong                                 |
+--------------------------------------------------------+-----------------------------------------+
```

Observe the URL in the first column entry, along with various literal types that follow. The URL is parsed to the new column verbatim, while the various literal types are all parsed as strings. All non-XMLSchema literal type column entries in the TSV-to-dataset translation will be formatted into the dataset as strings.

Here is an example of the same mixed-type column parsed as a `FloatType`:

```
+--------------------------------------------------------+-----------+
|?mixedtypes                                             |mixedtypes |
+--------------------------------------------------------+-----------+
|<http://www.Department10.University0.edu>               |null       |
|"5678"^^<http://www.w3.org/2001/XMLSchema#integer>      |null       |
|"5.678"^^<http://www.w3.org/2001/XMLSchema#float>       |5.678      |
|"tobeornot."^^<http://www.w3.org/2001/XMLSchema#string> |null       |
|"2001-99-99"^^<http://www.w3.org/2001/XMLSchema#date>   |null       |
|"somelong"^^<http://www.w3.org/2001/XMLSchema#long>     |null       |
+--------------------------------------------------------+-----------+
```

Note that the new `mixedtypes` column is intended for float values only, and only the one `XMLSchema#float` type is parsed to its base type (`float`), and all other entries of the resultant dataset column are set to null.

15.5.3 Role of the Spark Schema in TSV Translation

Spark programmers must write a schema to specify the TSV-to-dataset translation by the API. The API uses the schema to convert literal types from the TSV file into columns of their base types, such as integer, float, double, etc. The translated values are placed in columns in the resultant dataset with new column names as supplied by the schema, where any new column names supplied by the schema must differ from column names in the TSV file. If schema column name(s) are null or "", the API will derive new column names from the original TSV column names minus the first character. When TSV columnar data does not conform to the supplied schema, null values will be inserted in the translated column in the dataset. The number of `StructField` specifiers in the schema must equal the number of columns in the TSV file being parsed. The third argument to the `StructField` is always `true`, which allows for nullable column entries.

Sample Scala Schema

Below is a sample schema written in Scala.

```
//--- bring in necessary components for making schemas.
import org.apache.spark.sql.types._

//--- make a schema to use in directing the conversion of a tsv file to a dataset.
//--- String names for the schema StructFields (eg., "x-name") are used by
//--- the getSparkDatasetUsingSchema() API to name the new columns it will create when
//--- parsing XML literal types to base types. When "" or null are used in the schema,
//--- the API will derive the new column name from the tsv file column name.
val mySchema = StructType(Array(
                       StructField("x-theurl",StringType,true),
                       StructField("x-theinteger",IntegerType,true),
                       StructField("x-thefloat",FloatType,true),
                       StructField("",StringType,true),
                       StructField("x-thedate",DateType,true),
                       StructField(null,TimestampType,true),
                       StructField("x-thelong",LongType,true),
                       StructField("",DecimalType(38,10),true)
                       ))
```

This schema will direct the API to parse a TSV file to a Spark dataset that will consist of the following columns and their entry types, in the order shown:

- non-literal type, string column name: `x-theurl`
- integer literal-type, column name: `x-theinteger`
- float literal-type, column name: `x-thefloat`
- string, column name: blank, thus defaulting to the TSV column name. This column can be a literal-type as well.
- date literal-type, column name: `x-thedate`
- timestamp literal-type, null column name thus defaulting to the tsv column name

- long literal-type, column name: `x-thelong`
- decimal literal-type, column name: blank thus defaulting to the TSV column name

View the schema from a Spark Scala shell like this:

```
scala> mySchema.printTreeString
root
 |-- x-theurl: string (nullable = true)
 |-- x-theinteger: integer (nullable = true)
 |-- x-thefloat: float (nullable = true)
 |-- : string (nullable = true)
 |-- x-thedate: date (nullable = true)
 |-- null: timestamp (nullable = true)
 |-- x-thelong: long (nullable = true)
 |-- : decimal(38,10) (nullable = true)
```

Important points to note:

- if a column consists of mixed literal types, defining that column in the schema as "StringType" will result in all the literals in the column being parsed to their respective string values.
- decimal types are supported up to 38 digits of precision, with at most 10 digits to the right of the decimal point.

Sample Java Schema

Here is the same sample schema written in Java:

```
StructType customSchema = DataTypes.createStructType(new StructField[] {});
customSchema = customSchema.add("x-theurl",StringType,true);
customSchema = customSchema.add("x-theinteger",IntegerType,true);
customSchema = customSchema.add("x-thefloat",FloatType,true);
customSchema = customSchema.add("",StringType,true);
customSchema = customSchema.add("x-thedate",DateType,true);
customSchema = customSchema.add(null,TimestampType,true);
customSchema = customSchema.add("x-thelong",LongType,true);
customSchema = customSchema.add("",DecimalType(38,10),true;
```

View the schema from a Java program like this:

```
customSchema.printTreeString();
```

Sample Python Schema (Urika-GX ONLY)

Here is the same sample schema written in Python:

```
from py4j.java_gateway import java_import
from pyspark import SparkContext
from pyspark.sql.types import *
jvm = sc._gateway.jvm
java_import(jvm, "com.cray.cge.api.spark.SparkCgeApi.*")
.
.
.
jvm.com.cray.cge.api.spark.SparkCgeApi.makeNewSchemaTemplate()
jvm.com.cray.cge.api.spark.SparkCgeApi.addFieldToSchemaTemplate("x-theurl",StringType().simpleString(),True)
jvm.com.cray.cge.api.spark.SparkCgeApi.addFieldToSchemaTemplate("x-theinteger",IntegerType().simpleString(),True)
jvm.com.cray.cge.api.spark.SparkCgeApi.addFieldToSchemaTemplate("x-thefloat",FloatType().simpleString(),True)
jvm.com.cray.cge.api.spark.SparkCgeApi.addFieldToSchemaTemplate("",StringType().simpleString(),True)
jvm.com.cray.cge.api.spark.SparkCgeApi.addFieldToSchemaTemplate("x-thedate",DateType().simpleString(),True)
jvm.com.cray.cge.api.spark.SparkCgeApi.addFieldToSchemaTemplate("",TimestampType().simpleString(),True)
jvm.com.cray.cge.api.spark.SparkCgeApi.addFieldToSchemaTemplate("x-thelong",LongType().simpleString(),True)
jvm.com.cray.cge.api.spark.SparkCgeApi.addFieldToSchemaTemplate("",DecimalType(38,10).simpleString(),True)
```

View the schema from a Python program like this:

```
jvm.com.cray.cge.api.spark.SparkCgeApi.printSchemaTemplate()
```

Sample R Schema

Here is the same sample schema written in R:

```
library(SparkR)
sc <- sparkR.session(appName="SparkR-example")
.
.
.
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","makeNewSchemaTemplate")
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","addFieldToSchemaTemplate","x-theurl","STRING",TRUE)
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","addFieldToSchemaTemplate","x-theinteger","INT",TRUE)
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","addFieldToSchemaTemplate","x-thefloat","FLOAT",TRUE)
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","addFieldToSchemaTemplate","","STRING",TRUE)
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","addFieldToSchemaTemplate","x-thedate","DATE",TRUE)
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","addFieldToSchemaTemplate","","TIMESTAMP",TRUE)
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","addFieldToSchemaTemplate","x-thelong","BIGINT",TRUE)
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","addFieldToSchemaTemplate","","DECIMAL(38,10)",TRUE)
```

View the schema from an R program like this:

```
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","printSchemaTemplate")
```

Data types in Spark R

Data types in Spark R are identified using the strings, like `DATE`, `INT`, `FLOAT`, etc. This is because Spark R context (unlike Spark Scala, Python and Java) does not have built-in-type encoding that will yield those strings. Here is the complete set of available data types:

```
Spark Datatype         base type
--------------         ---------
"FLOAT" -------------- float
"STRING" ------------- string
"INT" ---------------- integer
"DATE" --------------- Date
"TIMESTAMP" ---------- Datetime
"BIGINT" ------------- long
"BINARY" ------------- long
"BOOLEAN" ------------ boolean
"TINYINT" ------------ byte
"DOUBLE" ------------- double
"SMALLINT" ----------- short
"DECIMAL(38,10)" ----- decimal
"TIME" --------------- Time
```

Scala and Java cases can use these strings as well, in place of the built-in-types shown in their cases. For example, the `FLOAT` string could replace the Scala or Java reference to `FloatType`, as well as the Python reference to `FloatType().simpleString()`.

Literal type to Data Types mapping

When writing a schema for parsing literal types to their base types, Spark programmers must use the correct Spark data type in the schema to match the literal type to parse for a given column. Here is the mapping:

```
"XMLSchema#"              Spark schema          Spark column      Spark R
literal type              DataType              primitive type    Datatype ID
------------              ------------          -------------     -----------
1) anyURI-----------------StringType------------String------------"String"
2) base64Binary-----------StringType------------String------------"String"
3) date-------------------DateType--------------Date--------------"DATE"
4) dateTime---------------TimestampType---------Timestamp---------"TIMESTAMP"
5) dateTimeStamp----------TimestampType---------Timestamp---------"TIMESTAMP"
6) dayTimeDuration--------StringType------------String------------"String"
7) decimal----------------DecimalType(38,10)----BigDecimal--------"DECIMAL(38,10)"
8) duration---------------StringType------------String------------"String"
9) ENTITIES---------------StringType------------String------------"String"
10) ENTITY----------------StringType------------String------------"String"
11) gDay------------------StringType------------String------------"String"
12) gMonth----------------StringType------------String------------"String"
13) gMonthDay-------------StringType------------String------------"String"
14) gYear-----------------StringType------------String------------"String"
```

```
15) gYearMonth------------StringType------------String------------"String"
16) hexBinary-------------BinaryType------------Long--------------"BINARY"
17) ID--------------------StringType------------String------------"String"
18) IDREF-----------------StringType------------String------------"String"
19) IDREFS----------------StringType------------String------------"String"
20) integer---------------IntegerType-----------Integer-----------"INT"
21) language--------------StringType------------String------------"String"
22) Name------------------StringType------------String------------"String"
23) NCName----------------StringType------------String------------"String"
24) negativeInteger-------IntegerType-----------Integer-----------"INT"
25) NMTOKEN---------------StringType------------String------------"String"
26) NMTOKENS--------------StringType------------String------------"String"
27) nonNegativeInteger----IntegerType-----------Integer-----------"INT"
28) nonPositiveInteger----IntegerType-----------Integer-----------"INT"
29) normalizedString------StringType------------String------------"String"
30) NOTATION--------------StringType------------String------------"String"
31) positiveInteger-------IntegerType-----------Integer-----------"INT"
32) QName-----------------StringType------------String------------"String"
33) time------------------TimestampType---------Time--------------"TIME"
34) token-----------------StringType------------String------------"String"
35) unsignedByte----------ByteType--------------Byte--------------"TINYINT"
36) unsignedInt-----------IntegerType-----------Integer-----------"INT"
37) unsignedLong----------LongType--------------Long--------------"BIGINT"
38) unsignedShort---------ShortType-------------Short-------------"SMALLINT"
39) boolean---------------BooleanType-----------Boolean-----------"BOOLEAN"
40) byte------------------ByteType--------------Byte--------------"TINYINT"
41) double----------------DoubleType------------Double------------"DOUBLE"
42) float-----------------FloatType-------------Float-------------"FLOAT"
43) int-------------------IntegerType-----------Integer-----------"INT"
44) long------------------LongType--------------Long--------------"BIGINT"
45) short-----------------ShortType-------------Short-------------"SMALLINT"
46) string----------------StringType------------String------------"String"
47) yearMonthDuration-----StringType------------String------------"String"
```

The first column shows the possible literal types that can appear in CGE TSV results files. The second column shows the data types that can be written into Spark schema's for the Scala, Java, and Python cases when converting TSV files to Spark datasets. The third column shows the base-type (aka., "primitive data type") used in resultant Spark dataset columns that are created when the API parses the given literal type (first column) using the schema data type (in the second column) into a Spark dataset column (third column). The fourth column shows the data types that can be written into Spark schema's for the R case.

15.5.4 Example of Spark Scala to Spark Dataset Conversion

While the full range of Spark operations on datasets is beyond the scope of this section, here are a few examples of the simpler operations Spark programmers can use.

Following is a simple Spark Scala program to convert a two column TSV file to a dataset, and execute simple Spark commands on the dataset:

```
//--- Produce an instance of the "Hello from CGE Java API!" return text,
val the_str = com.cray.cge.api.spark.SparkCgeApi.helloWorld(false)

//--- bring in necessary components for making schemas.
import org.apache.spark.sql.types._

//--- make a schema to use in directing the conversion of a tsv file to a Spark dataset.
val schema3 = StructType(Array(
                       StructField("x-theurl",StringType,true),
                       StructField("x-theinteger",IntegerType,true)
                       ))
// look at the schema
schema3.printTreeString

//--- Run the API to parse a tsv file and create new columns according to the schema.
val retainOrigCols = false
val this_df_2 = com.cray.cge.api.spark.SparkCgeApi.getSparkDatasetUsingSchema("file:///queryResults.
2016-11-02T16.45.33Z000.12520.tsv", schema3, retainOrigCols)

//--- these statements show that we converted the CGE tsv file to a Spark dataset
if (this_df_2 != null) {
   //--- look at the schema of the resultant dataset, count number of rows, and dump the first 2 rows.
   this_df_2.printSchema()
   this_df_2.count()
   this_df_2.show(2,false)

   //--- isolate and display the columns of the resultant dataset: the original tsv-file column + parsed column.
   if (retainOrigCols) {
      val urlcols = this_df_2.select("?type","x-theurl")
      urlcols.show(99,false)
      val intcols = this_df_2.select("?usages","x-theinteger")
      intcols.show(99,false)
   } else {
      //--- isolate and display the columns of the resultant dataset.
      val urlcols = this_df_2.select("x-theurl")
      urlcols.show(99,false)
      val intcols = this_df_2.select("x-theinteger")
      intcols.show(99,false)
   }
}
```

In the following, `queryResults.2016-11-02T16.45.33Z000.12520.tsv` in local file space is being converted to dataset:

```
?type                                                ?usages
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag>     "11"^^<http://www.w3.org/2001/XMLSchema#integer>
<http://localhost/vocabulary/bench/Article>          "2077"^^<http://www.w3.org/2001/XMLSchema#integer>
<http://localhost/vocabulary/bench/Inproceedings>    "621"^^<http://www.w3.org/2001/XMLSchema#integer>
<http://localhost/vocabulary/bench/Incollection>     "33"^^<http://www.w3.org/2001/XMLSchema#integer>
<http://localhost/vocabulary/bench/Proceedings>      "18"^^<http://www.w3.org/2001/XMLSchema#integer>
<http://xmlns.com/foaf/0.1/Document>                 "2806"^^<http://www.w3.org/2001/XMLSchema#integer>
<http://localhost/vocabulary/bench/Journal>          "56"^^<http://www.w3.org/2001/XMLSchema#integer>
<http://xmlns.com/foaf/0.1/Person>                   "2162"^^<http://www.w3.org/2001/XMLSchema#integer>
```

Following are the outputs of running the code:

```
[user@nid00030 username]$ spark-shell --jars /home/users/$USER/cge-java-api-v1.3.0-with-dependencies.jar --conf
"spark.debug.maxToStringFields=38"

Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
.
.
.
Spark context Web UI available at http://192.168.0.31:4040
Spark context available as 'sc' (master = mesos://zk://zoo1:2181,zoo2:2181,zoo3:2181/mesos, app id = f55c6778-
a4ce-4995-9e1f-daf47efb9d37-0034).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_111)
Type in expressions to have them evaluated.
Type :help for more information.

scala>

scala>

scala> :load simple.scala
Loading simple.scala...

the_str: String = SparkCgeApi.helloWorld - Hello from Java CGE Spark API!

import org.apache.spark.sql.types._
schema3: org.apache.spark.sql.types.StructType = StructType(
         StructField(x-theurl,StringType,true),
         StructField(x-theinteger,IntegerType,true
         ))

root
 |-- x-theurl: string (nullable = true)
 |-- x-theinteger: integer (nullable = true)

retainOrigCols: Boolean = false
this_df_2: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [x-theurl: string, x-theinteger: int]

root
 |-- x-theurl: string (nullable = true)
 |-- x-theinteger: integer (nullable = true)

+------------------------------------------------+------------+
|x-theurl                                        |x-theinteger|
+------------------------------------------------+------------+
|<http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag>|11          |
|<http://localhost/vocabulary/bench/Article>     |2077        |
+------------------------------------------------+------------+
only showing top 2 rows

+-------------------------------------------------+
|x-theurl                                         |
+-------------------------------------------------+
|<http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag> |
|<http://localhost/vocabulary/bench/Article>      |
|<http://localhost/vocabulary/bench/Inproceedings>|
|<http://localhost/vocabulary/bench/Incollection> |
|<http://localhost/vocabulary/bench/Proceedings>  |
|<http://xmlns.com/foaf/0.1/Document>             |
|<http://localhost/vocabulary/bench/Journal>      |
|<http://xmlns.com/foaf/0.1/Person>               |
+-------------------------------------------------+

+------------+
|x-theinteger|
+------------+
|11          |
|2077        |
|621         |
|33          |
|18          |
|2806        |
|56          |
|2162        |
+------------+
```

Following is an example of a Spark Scala `select` and `show` statements on a dataset containing a float type column named `x-thefloat` in the schema, along with the original column of literal-type floats that appeared as `?thefloat` column in the TSV file and was brought into the resultant dataset by setting the `retainOriginalColumn` argument to `true`:

```
val floatcols = df.select("?thefloat","x-thefloat")
floatcols.show(8,false)

+---------------------------------------------------------+-------------+
|?thefloat                                                |x-thefloat   |
+---------------------------------------------------------+-------------+
|"9.1"^^<http://www.w3.org/2001/XMLSchema#float>          |9.1          |
|"-3E2"#misformed_float>                                  |null         |
|"4268.22752E11"^^<http://www.w3.org/2001/XMLSchema#float>|4.26822743E14|
|"+24.3e-3"^^<http://www.w3.org/2001/XMLSchema#float>     |0.0243       |
|"+5.5"^^<http://www.w3.org/2001/XMLSchema#float>         |5.5          |
|"-INF"^^<http://www.w3.org/2001/XMLSchema#float>         |-Infinity    |
|"NaN"^^<http://www.w3.org/2001/XMLSchema#float>          |NaN          |
|"-0.123"^^<http://www.w3.org/2001/XMLSchema#float>       |-0.123       |
+---------------------------------------------------------+-------------+
```

The misformed float literal type in the `?thefloat` column of the original TSV file adds a `null` entry in the `x-thefloat` column

15.5.5 Errors and Exceptions Encountered while Using the CGE Spark API

Following are a list of potential errors and exceptions the Spark programmer may encounter when working with the CGE Spark API.

Errors

Most errors will occur when specifying a schema that does not match the number of columns or types in the TSV file to convert:

- Mistakenly parsing a StringType column as any numeric type (IntegerType, DecimalType, etc.) will result in the column entries coming back null.

- Specifying more columns in the schema than exist in the TSV file will result in the API giving a message "ERROR! df column count does not match schema column count"
- Attempting to convert a TSV file that does not exist will result in an error message "SparkCgeApi.getSparkDataset[UsingSchema] - error with file ... java.lang.NullPointerException"
- Attempting to execute the Spark select command on non-existing column will produce an error "org.apache.spark.sql.AnalysisException: cannot resolve '`*column_name*`'"
- Specifying a column name in a schema that exactly matches the same column name in the TSV file will result in that column's literal type not being parsed into the resultant dataset. Subsequent attempts to do spark "select" commands on that column name will result in the org.apache.spark.sql.AnalysisException described earlier.

Exceptions

A general rule for some numeric types: if the literal value is out range, a Java exception may be thrown. For example, if a column element "0"^^<http://www.w3.org/2001/XMLSchema#negativeInteger> is present in a TSV file, it will result in the Java exception "Lexical form '0' is not a legal instance of Datatype[http://www.w3.org/2001/XMLSchema#negativeInteger]".

Another general rule for some numeric types is that if the literal value is out of range, the resultant column element value will be the minimum for the type. For example, if a column element "0xAB"^^<http://www.w3.org/2001/XMLSchema#long> is present in a TSV file, the resultant parsed column value will be -9223372036854775808, the minimum for "long" types. For out of range "integer" types, the resultant column value will be -2147483648.

15.5.6 Run CGE from Spark

This feature is currently supported on Urika-GX only.

Spark programmers can start, stop, and run queries on CGE from Spark context. The following are code samples showing how to do this.

Spark Scala

```
// Start CGE

// Any valid port number
val cge_port = 23239

// Path to dataset to start CGE on
val dataset_dir = "/mnt/lustre/ripple/mkdb_1.0/lubm/0"

// Path to directory CGE should send results
val output_dir = "./"

// CGE log filenames
val database_log = "database_log.txt"
val launcher_log = "launcher_log.txt"

// number of nodes and images per node to start CGE on
val node_count = 4
val image_count = 2
```

```
// Timeouts
val run_time_min = 3
val startup_timeout_sec = 20

val show_progress = true
val started = com.cray.cge.api.spark.SparkCgeApi.startCgeServer(cge_port,
dataset_dir, output_dir, database_log, launcher_log, node_count, image_count,
run_time_min, startup_timeout_sec, show_progress)

// Send CGE a query and receive the output tsv file
val the_query = "SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?type } GROUP
BY ?type"
val tsv_results = com.cray.cge.api.spark.SparkCgeApi.queryRunningCGE(cge_port,
the_query, show_progress)

// Convert the tsv file output to a Spark dataset
val df = com.cray.cge.api.spark.SparkCgeApi.getSparkDataset(tsv_results,
show_progress)

// work with the new dataset
if (df != null) {
   df.printSchema()
   df.count()
}

// Stop CGE
val connection_timeout_sec = 20
com.cray.cge.api.spark.SparkCgeApi.stopCge(cge_port, connection_timeout_sec)
```

Spark Java

```
package com.cge.spark.api.test.app;
import org.apache.spark.sql.*;
import com.cray.cge.api.spark.*;
.
.
.
// make an instance of the SparkCgeApi.
SparkCgeApi the_api = new SparkCgeApi();

// Start CGE
int cge_port = 22334;
boolean started = false;
int runtime = 3;           // CGE runtime limit, minutes
int startup_timeout = 20;  // CGE Startup timeout limit, seconds.
String launcher_log = "launcher_log.txt";
String database_log = "database_log.txt";
val show_progress = true;
started = the_api.startCgeServer(cge_port, dataset_dir, output_dir, database_log,
launcher_log, node_count, image_count, runtime, startup_timeout, true);

if (started) {
   // Run a query on CGE
   String DEFAULT_QUERY = "SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?
type } GROUP BY ?type";
   String tsv_results = the_api.queryRunningCGE(cge_port, DEFAULT_QUERY,
show_progress);
   Dataset<Row> df = the_api.getSparkDataset(tsv_results, show_progress);
   // work with the new dataset
   if (df) {
      df.printSchema();
      df.count()
   }

   // Stop CGE
   int connect_timeout = 20;  // connection timeout, seconds.
   the_api.stopCge(cge_port, connect_timeout);
}
```

Spark Python

```
from py4j.java_gateway import java_import
from pyspark import SparkContext
# sc = SparkContext(appName="my_python")
jvm = sc._gateway.jvm
java_import(jvm, "com.cray.cge.api.spark.SparkCgeApi.*")
#--- simple test of the SparkCgeApi - returns a "hello world" string.
#--- (note the "True/False" argument to 'helloWorld' enables banners showing
progress in the Spark CGE API)
the_str = jvm.com.cray.cge.api.spark.SparkCgeApi.helloWorld(False)
print(the_str)
#--- Test API by starting CGE
cge_port = 23239
dataset_dir = "/mnt/lustre/ripple/mkdb_1.0/lubm/0"
output_dir = "./"
database_log = "database_log.txt"
launcher_log = "launcher_log.txt"
node_count = 1
image_count = 2
run_time_min = 3
startup_timeout_sec = 30
started =
jvm.com.cray.cge.api.spark.SparkCgeApi.startCgeServer(cge_port,dataset_dir,output_
dir,database_log,launcher_log,node_count,image_count,run_time_min,startup_timeout_
sec,True)
print(started)
#--- setup a local tsv file (ok to overwrite with newer file below)
tsv_results = "file:///some_path/queryResults.2017-05-08T19.58.19Z000.66015.tsv"
if started:
   #--- Test API by sending query to the running CGE, get back path to tsv
results file ("True" = show progress).
   the_query = "SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?type } GROUP
BY ?type"
   tsv_results = jvm.com.cray.cge.api.spark.SparkCgeApi.queryRunningCGE(cge_port,
the_query, True)
   print(tsv_results)
   #
   #--- Test API by converting the tsv file from above to a Spark dataset ("True"
= show progress).
   ds = jvm.com.cray.cge.api.spark.SparkCgeApi.getSparkDataset(tsv_results,True)
   #
   #--- take a look at the resultant dataset
   if ds != None:
      ds.printSchema()
      ds.count()
      ds.show(4,False)
   #
   #--- Test API by stopping CGE
   connection_timeout = 30
   jvm.com.cray.cge.api.spark.SparkCgeApi.stopCge(cge_port, connection_timeout)
```

Spark R

```
library(SparkR)
sc <- sparkR.session(appName="SparkR-example")
#--- simple test of the SparkCgeApi
test_string =
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","helloWorld",TRUE)
print(test_string)
#--- setup a local tsv file (ok to overwrite with newer file below)
local_tsv_file = "file:///home/users/schema_test/test/Rtest/queryResults.
2017-05-08T19.58.19Z000.66015.tsv"
#--- Test API by starting CGE
#--- note the "L" qualifyer on numbers is important here in R context - won't be
seen as an
#--- integer in the JVM without it!
cge_port = 23239L
dataset_dir = "/mnt/lustre/ripple/mkdb_1.0/lubm/0"
output_dir = "./"
database_log = "database_log.txt"
launcher_log = "launcher_log.txt"
node_count = 1L
image_count = 2L
run_time_min = 3L
startup_timeout_sec = 30L
started =
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","startCgeServer",cge_port,
dataset_dir,output_dir,database_log,launcher_log,node_count,image_count,run_time_m
in,startup_timeout_sec,TRUE)
print(started)
if (started) {
   #--- Test API by sending a query to an already running CGE. Convert the
resultant tsv file to a dataframe.
   #--- (note the "TRUE/FALSE" argument should enable banners showing progress in
the API)
   the_query = "SELECT ?type (COUNT(?s) AS ?usages) WHERE { ?s a ?type } GROUP
BY ?type"
   print(the_query)
   local_tsv_file =
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","queryRunningCGE",cge_port
,the_query,FALSE)
   print(local_tsv_file)
   #--- Test API by converting the CGE tsv results file to a Spark dataset.
   ds_1 =
sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","getSparkDataset",local_ts
v_file, TRUE)
   #--- peculiar step you have to take in R to work with datasets.
   df_1 <- new("SparkDataFrame", ds_1, FALSE)
   #--- these statements show that we got the CGE tsv output and converted it to
a Spark 'R' dataframe
   if (!is.null(df_1)) {
      printSchema(df_1)
      count(df_1)
      showDF(df_1, numRows = 4, FALSE)
   }
```

```
    #--- Test API by stopping CGE
    connection_timeout = 20L

sparkR.callJStatic("com.cray.cge.api.spark.SparkCgeApi","stopCge",cge_port,connect
ion_timeout)
}
```

15.5.7 CGE Spark DataFrame to RDF Triples Data Converter

Scala and Java methods can be used to write RDF files from Spark. The Spark user would create a three-column dataframe to pass to the dataframe-to-RDF converter. These columns would, respectively, represent the subject, predicate and object fields of the RDF triples to be generated. In particular, `saveAsRDF(name: DataFrame, path: String, objectIsURI: Boolean)` when called from Spark will write one or more files containing triples in RDF format from an existing dataframe. Name is the name of the `DataFrame` which is assumed to have been built in advance. The path specifies the directory the files are to be written to. For non HDFS the path must begin with "`file://<path_name>`". `ObjectIsURI` is a flag that controls how the user wants to modify the object. The dataframe must contain values in the first three columns that conform to the specified criteria; successive columns are ignored. The first three columns correspond to Subject, Predicate and Object, respectively.

It is anticipated that the `DataFrame` to be written out will have been created in advance. The `saveAsRDF()` method will write two or more files to the directory specified by path. The resulting files found in the specified directory is a single `graph.info` file along with one or more RDF triples files. The `graph.info` file lists all files constituting the set of output triples. CGE can read the `graph.info` file and its accompanying files that it lists and build a CGE internal database, dbQuads, from it. The subject and predicate columns are expected to be strings.

- if the string looked like a URL, i.e., starts with "`//http:`" etc., it will be converted into a URI, simply by enclosing it in angle brackets "`<...>`".
- if the string looked like any other type of string, such as a variable name, the sting would be converted by prepending "`urn:`" and enclosing the resulting string in angle brackets.

The object column could be a string, or it could contain one of the basic data types that can be stored in a dataframe: integer, float, double, and so on. If the object is a numerical data type, it is to be converted to the XMLSchema equivalent string. For example, the integer 439 would be converted to "`439"^^<http://www.w3.org/2001/XMLSchema#integer>` . Information about the data type of the column would be expected to be found in the schema associated with the dataframe.

If the object column is a string, it can be handled either of two ways:

- The user wants it treated as a URI, in which case the column is treated the same way as the subject and predicate columns are, as described above.
- The user wants it treated a string literal of data type XMLSchema string, in which case, for example, the string `xxyyzz` would be converted to the XLMSchema literal "`xxyyzz"^^<http://www.w3.org/2001/XMLSchema#string>` . Since there is no way for the conversion interface code to tell from the dataframe schema, which of these choices the user desires, the user will need to state their preference via the Boolean argument, the `ObjectIsURI` flag, to the dataframe-to-RDF conversion function.

If `saveAsRDF()` succeeds it returns 0; otherwise a negative integer indicating an error code is returned. At present the only error code is -1, signifying that an unexpected type was detected.

Usage

Because `saveAsRDF()` is a member function of the class `rdfWriter` it is necessary to first create an `rdfWriter` object before using its method. For example:

```
val tempWriter = new rdfWriter; val result = tempWriter.saveAsRDF(thisDF, "file:///mnt/lustre/myFile", false)
```

Limitations

Because the datatype is specified per column in the schema, it is not possible to write out triples that have non-uniform object datatypes. Currently, handling object literals of the following types:

- `DateTimeDuration`
- `Date`
- `Time`
- `DateTime`
- `TypedString`
- `StringLang` or unsigned numerics, including `Unsigned Long`, `Unsigned Int`, `Unsigned Short`, `Unsigned Byte`

.

16 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`

16.1 CGE Error Messages and Resolution Information

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. Common error messages that are likely to be encountered along with troubleshooting techniques are documented in the following table.

Table 20. 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 Directory`	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.
`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. This may also be caused by the CLI selecting the wrong SSH identity. As described in the SSH identities section, the first identity found by searching several default locations is used, but this may not always be the desired identity.	● 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 ○ 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.

Error Message	Description	Resolution
		● Use the `--identity` option to specify the desired identity directory to use
`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 a longer period of time. ○ Disable the timeout by setting `--timeout 0` to wait indefinitely until the

Error Message	Description	Resolution
		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
`Server failed to start up`	One or more of the CGE job steps failed to launch because CGE was not found.	Try relaunching CGE if the system displays this message. In addition, it is recommended to ensure that all compute nodes are correctly configured. In particular verify the following: ● The same version of CGE is installed on all compute nodes and the login nodes ● All shared file systems are mounted and mounted in the same place on all compute nodes and the login nodes ● The munged process is running on all compute nodes If any of the preceding is not true and if relaunching the CGE CLI does not correct the problem, contact Cray Support.
`Not enough symmetric heap for new sorting keys`	There is not enough symmetric heap for new sorting keys	use the `-H` option to `cge-launch` to set the symmetric heap value to a larger value. Try doubling what shows up by default near the top of the log for a start. Symmetric heap is a boundary value on a resource that is allocated as needed, so using a larger than necessary value does not mean that this value will be allocated. It only means that no more than this value will be allocated. It is better to overestimate by a bit than to underestimate.

Error Message	Description	Resolution
`[PE_64]:inet_listen_socket_setup :inet_setup_listen_socket: bind failed port 20219 listen_sock = 5 Address already in use`	This may be due to leftover `cge-server` processes	Follow the instructions documented in *Terminate Orphaned cge-server Jobs* on page 191
`Error: Timed out waiting for the server to start running`	When a computational loop during a database build takes an extremely long time without producing any indication of forward progress (generally some kind of output in the log), `cge-launch` may decide that the start up sequence has hung and terminate it with this message.	Change the interval used to detect a start up hang from its default setting of 900 seconds to some longer interval. If you know the problem is just that a dataset is very computationally intensive to build and is prone to such timeouts, setting this timeout value to 3600 seconds (an hour) is almost certain to eliminate any chance of this failure at the expense of causing you to take a very long time to detect an actual hang in start up. To change this, use the `--startupTimeout=`*`seconds`* option to `cge-launch`.
`HTTP Errors are reported by a tool or API`	A request submitted to the HTTP Interface provided by the `cge-cli fe` command was not successful. If the request was submitted via a tool or API then only minimal error details may be reported directly to you. However please see the resolutions for ways to find more detailed error information.	● Submit the same request using a browser. The browser window may contain additional error messages which indicate the underlying error. Please review these carefully since they may indicate one of the other common errors detailed in this table. ● Please review the front end logging as this will have logged the HTTP error and associated error details. These may indicate one of the other common errors detailed in this table. ● If there is no obvious cause or additional error messages in the browser/front end logs then please review the database logs for error messages that may indicate if/why the request failed on the database server. ● In rare cases, the offending request may have caused the database server to crash in

Error Message	Description	Resolution
		which case, it will be necessary to relaunch it before making further requests ○ If a crash has occurred please report this to your Cray support representative
`:inet_listen_socket_setup :inet_setup_listen_socket : bind failed port 1371 listen_sock = 5 Address already in use`	A previous `cge-launch` or HPC/`mrun` job failed or was killed, and the inet_listen socket is likely in the `TIME_WAIT` state on one or more of the compute nodes.	Wait 60-90 seconds for the `inet_listen_socket` (port 1371) to clear up from `TIME_WAIT` state. If the problem persists, the likely cause is some other program has an active socket connection to port 1371 on one (or more) compute nodes. That application must release port 1371 on the affected node(s) before new `cge-launch` or HPC/`mrun` jobs can be run on that node(s).
`User user does not have permission to perform operation` *`operation`*	An action was requested for which the requesting user did not have the appropriate permissions	● Submit the request as a user who does have the appropriate permissions ● Contact the database owner and ask if you can be granted the appropriate permissions

16.2 Terminate Orphaned cge-server Jobs

Prerequisites

This procedure requires root privileges.

About this task

Follow the instructions listed in this procedure to track orphaned `cge-server` jobs down and terminate them. The examples shown in this procedure can be used for a system with 3 sub-racks.

Procedure

1. Log on to the System Management Workstation (SMW) as root
2. Execute the following to find out if there are stray `cge-server` processes.

   ```
   # pdsh -w 'nid000[00-47]' "ps -ef|grep 'cge-serve[r]'|grep -v grep | awk '{print \$2}';true"|wc -w
   ```

3. Terminate the stray `cge-server` processes

```
# pdsh -w nid000[00-30,32-46] "ps -ef|grep 'cge-serve[r]'|awk '{print \$2}'|xargs kill"
```

4. Rerun the preceding command to ensure all stray `cge-server` processes have been terminated.

5. Verify that all the stray `cge-server` processes have been terminated by executing the following command:

```
# pdsh -w 'nid000[00-47]' "ps -ef|grep 'cge-serve[r]'|grep -v grep | awk '{print \$2}';true"|wc -w
0
```

This output indicates that everything has been cleared.

16.3 Diagnose CGE Python API Issues

Exceptions

The Java JVM will pass exception information back to the python interpreter. Here are examples of common runtime and programming errors that produce exceptions:

- **Starting CGE with a reference to a nonexistent dataset** - An exception will occur if the dataset referenced in the `forExistingDatabase()` invocation does not exist.

```
>>>
>>> my_cge_launcher_builder.forExistingDatabase("/mnt/lustre/xxx/ripple/mkdb/sp2b/25k")

Traceback (most recent call last):
  File "test.py", line 66, in <module>
    my_cge_launcher_builder.forExistingDatabase("/mnt/lustre/xxx/ripple/mkdb/sp2b/25k")
  File "/usr/lib/python2.7/site-packages/py4j/java_gateway.py", line 1133, in __call__
    answer, self.gateway_client, self.target_id, self.name)
  File "/usr/lib/python2.7/site-packages/py4j/protocol.py", line 319, in get_return_value
    format(target_id, ".", name), value)
py4j.protocol.Py4JJavaError: An error occurred while calling o6.forExistingDatabase.
: java.lang.IllegalArgumentException: Database directory /mnt/lustre/xxx/ripple/mkdb/sp2b/25k must be an existing directory
        at com.cray.cge.api.builders.CgeLauncherBuilder.forExistingDatabase(CgeLauncherBuilder.java:65)
        at com.cray.cge.api.builders.CgeLauncherBuilder.forExistingDatabase(CgeLauncherBuilder.java:95)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:237)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
        at py4j.Gateway.invoke(Gateway.java:280)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.GatewayConnection.run(GatewayConnection.java:214)
        at java.lang.Thread.run(Thread.java:745)
```

- **Running a query against a connection where the cge-server has already exited** - The `my_conn` object is still valid, but the call to `querySummary()` generates an exception because the CGE server is not running.

```
>>> my_conn.isRunning()
False
>>>
>>>
>>> my_query_results = my_conn.querySummary(DEFAULT_QUERY)
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
   File "/usr/lib/python2.7/site-packages/py4j/java_gateway.py", line 1133, in __call__
     answer, self.gateway_client, self.target_id, self.name)
   File "/usr/lib/python2.7/site-packages/py4j/protocol.py", line 319, in get_return_value
     format(target_id, ".", name), value)
py4j.protocol.Py4JJavaError: An error occurred while calling o25.querySummary.
: com.hp.hpl.jena.query.QueryExecException: There was an error communicating with the remote server
         at com.cray.cge.sparql.engine.CgeQueryEngine.eval(CgeQueryEngine.java:157)
         at com.hp.hpl.jena.sparql.engine.QueryEngineBase.evaluateNoMgt(QueryEngineBase.java:142)
         at com.hp.hpl.jena.sparql.engine.QueryEngineBase.createPlan(QueryEngineBase.java:110)
         at com.hp.hpl.jena.sparql.engine.QueryEngineBase.getPlan(QueryEngineBase.java:88)
         at com.cray.cge.api.builders.CgeConnectionImpl.querySummary(CgeConnectionImpl.java:628)
         at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
         at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
         at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
         at java.lang.reflect.Method.invoke(Method.java:498)
         at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:237)
         at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
         at py4j.Gateway.invoke(Gateway.java:280)
         at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
         at py4j.commands.CallCommand.execute(CallCommand.java:79)
         at py4j.GatewayConnection.run(GatewayConnection.java:214)
         at java.lang.Thread.run(Thread.java:745)
Caused by: com.cray.cge.communications.messaging.exceptions.CommunicationsSecurityException: \
Unable to establish a connection to the database server at localhost:23239 as it does not appear to be running
         at com.cray.cge.communications.client.ssh.SshClient.connect(SshClient.java:484)
         at com.cray.cge.communications.client.AbstractClient.connect(AbstractClient.java:61)
         at com.cray.cge.sparql.engine.CgeQueryEngine.eval(CgeQueryEngine.java:102)
         ... 15 more Caused by: com.jcraft.jsch.JSchException: java.net.ConnectException: Connection refused
         at com.jcraft.jsch.Util.createSocket(Util.java:394)
         at com.jcraft.jsch.Session.connect(Session.java:215)
         at com.cray.cge.communications.client.ssh.SshClient.connect(SshClient.java:439)
         ... 17 more Caused by: java.net.ConnectException: Connection refused
         at java.net.PlainSocketImpl.socketConnect(Native Method)
         at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
         at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
         at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
         at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
         at java.net.Socket.connect(Socket.java:589)
         at java.net.Socket.connect(Socket.java:538)
         at java.net.Socket.<init>(Socket.java:434)
         at java.net.Socket.<init>(Socket.java:211)
         at com.jcraft.jsch.Util$1.run(Util.java:362)
```

- **Invoking `withJobOptions()` more than once -** This shows how the `withJobOptions()` function can only be invoked once for a given instance of the `CgeLauncherBuilder`.

```
>>>
>>> my_cge_launcher_builder.withJobOptions(my_cge_joboptions)
>>>
>>> my_cge_launcher_builder.withJobOptions(my_cge_joboptions)
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
   File "/usr/lib/python2.7/site-packages/py4j/java_gateway.py", line 1133, in __call__
     answer, self.gateway_client, self.target_id, self.name)
   File "/usr/lib/python2.7/site-packages/py4j/protocol.py", line 319, in get_return_value
     format(target_id, ".", name), value)
py4j.protocol.Py4JJavaError: An error occurred while calling o7.withJobOptions.
: java.lang.IllegalStateException: Cannot set job options as they have already been set
         at com.cray.cge.api.builders.CgeLauncherBuilder.withJobOptions(CgeLauncherBuilder.java:144)
         at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
         at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
         at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
         at java.lang.reflect.Method.invoke(Method.java:498)
         at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:237)
         at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
         at py4j.Gateway.invoke(Gateway.java:280)
         at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
         at py4j.commands.CallCommand.execute(CallCommand.java:79)
         at py4j.GatewayConnection.run(GatewayConnection.java:214)
         at java.lang.Thread.run(Thread.java:745)
```

Errors

- **Attempt to access gateway that has been shutdown -** This error shows a legitimate shutdown of the JVM, but then an attempt to utilize the previously active connection.

```
>>>
>>> gateway.shutdown()
>>>
>>> my_conn.getPort()
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
   File "py4j/java_gateway.py", line 1131, in __call__
     answer = self.gateway_client.send_command(command)
   File "py4j/java_gateway.py", line 881, in send_command
     connection = self._get_connection()
   File "py4j/java_gateway.py", line 825, in _get_connection
     raise Py4JNetworkError("Gateway is not connected.")
py4j.protocol.Py4JNetworkError: Gateway is not connected.
>>>
>>>
```

- **Shutting down the gateway before stopping the connection**- This error shows a legitimate shutdown of the JVM, then an attempt to stop the CGE server.

  ```
  >>>
  >>> gateway.shutdown()
  >>>
  >>> my_conn.stop()
  Traceback (most recent call last):
     File "<stdin>", line 1, in <module>
     File "py4j/java_gateway.py", line 1131, in __call__
       answer = self.gateway_client.send_command(command)
     File "py4j/java_gateway.py", line 881, in send_command
       connection = self._get_connection()
     File "py4j/java_gateway.py", line 825, in _get_connection
       raise Py4JNetworkError("Gateway is not connected.")
  py4j.protocol.Py4JNetworkError: Gateway is not connected.
  ```

- **Not enough CPUs available to launch CGE** - After starting the connection and waiting a suitable start up time, the call to `isRunning()` returns `False`, and the call for `status()` returns `Failed` and `NotRunning`.

  ```
  >>> my_conn.start()
  >>>
  >>> my_conn.isRunning()
  False
  >>>
  >>> my_CgeStatus = my_conn.status()
  >>> my_CgeStatus.toString()
  u'Process: Failed - CGE: NotRunning'
  ```

  The error can be seen in the `cge_runtime.log`.

  ```
  Tue Sep 20 2016 16:28:38.336870 CDT[][mrun]:ERROR:Not enough CPUs for exclusive access. Available: 1 Needed: 2
  ```

- **Exiting python without explicitly running `gateway.shutdown()`** - This leaves the Java JVM process as a still-active orphan process.

  ```
  [userid@nid00030 ~]$ top -u $USER
    PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM   TIME+   COMMAND
  64461 userid    20   0 35.778g  36304  14640 S   0.0  0.0   0:00.42 java
  ```

  in which case the user should kill the process explicitly:

  ```
  [userid@nid00030~]$ kill -964461
  ```