Ryft REST Command

User & Admin Guide

Ryft Document Number: 1191

Ryft REST Release Number: 0.8.1

Document Version: 1.1.0

Revision Date: November 2016

Revision History

Date	Reason for Change	Version
Oct. 2016	Initial Release	1.0.0
Nov. 2016	Update API endpoints and examples	1.1.0

Contents

Using This Guide

This guide is for the user who will use the Ryft ONE REST API to access the Ryft ONE and use its primitives.

Additional resources: Ryft Open API Library User Guide and the *Ryft ONE: User Guide*.

Ryft Technical Support

You may access our first-tier support directly from our public website by using the Chat Widget and starting a chat with our support agents. All chat conversations are tracked and become help desk tickets in our support system. In addition, you can access our support system through the Ryft Support site: https://support.ryft.com. Log in with your credentials to see your past tickets, create new tickets, and to access limited-access content.

For technical support, contact us:

Email: support@ryft.com

Web: https://support.ryft.com

Phone: 1-855-793-8663 (RYFT ONE)

About Ryft Systems Inc.

Ryft is the performance leader in data-intensive computing with the first commercial HPC appliance to use hybrid x86/FPGA compute, 48 TB of storage, and an Open API to accelerate and streamline existing software ecosystems by 100X.

With more than a decade of experience delivering incredibly fast data analysis solutions to government intelligence agencies, Ryft is the only company that understands how to effectively apply the combination of FPGA compute acceleration and x86 integration to a broad set of data-intensive workloads from the Internet of Things, video cameras, web logs, customer data, and other sources. Industry heavyweights in retail, finance, defense, and health care trust Ryft to power a range of real-time intelligence applications.

1

1: Overview

The Ryft RESTful JSON server (`ryft-server`) runs as a daemon and provides access to the Ryft hardware via the `libryftone` library or the `ryftprim` command line tool. It is written in Go using Gin HTTP framework. The server contains a *search engine abstraction* which unifies access to the RyftONE primitive library and is a natural fit to operating in a Ryft Cluster.

The search engine interface looks like this:

```
type Engine interface {
    Search(cfg *Config) (*Result, error)
    Count(cfg *Config) (*Result, error)
    Files(dir string) (*DirInfo, error)

    Options() map[string]interface{}
}
```

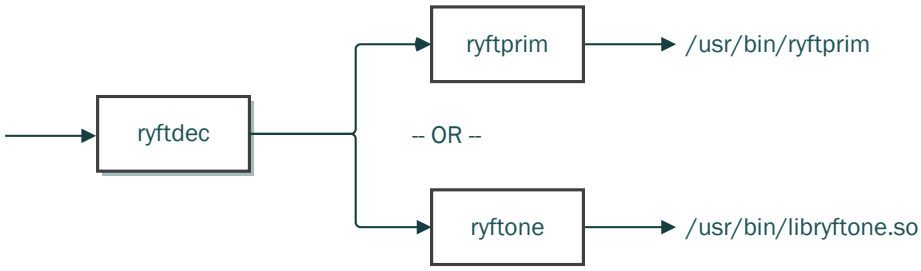
Most of the interface methods are related to the corresponding REST API endpoints: `/search`, `/count` and `/files`. The `Options()` method is used to get search engine's internal options which are, initially, customizable via the search configuration file.

The `ryft-server` uses this abstract search engine in its main code so that we can easily change the actual search engine implementation. For example, for test purposes it's quite easy to create a fake search engine which uses a simple `grep` tool on the local filesystem.

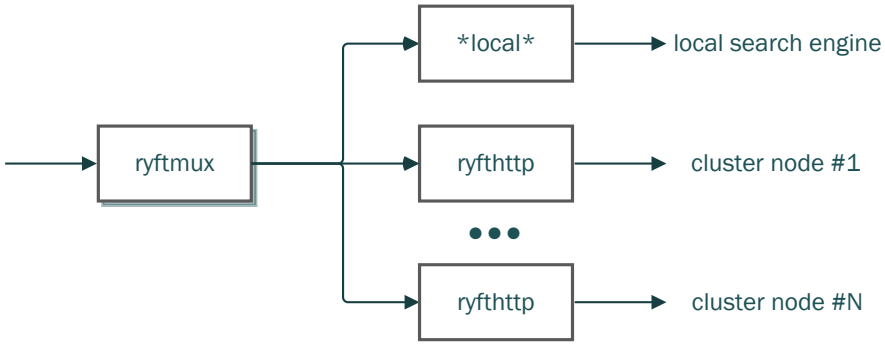
This is a list of search engine implementations:

- ryftprim uses the `ryftprim` command line tool
- ryftone uses the `libryftone` library from Ryft Open API
- ryfthttp uses another `ryft-server` instance
- ryftmux multiplexes results from several search engines
- ryftdec decomposes complex search queries

The `ryftprim` and `ryftone` search engines are used for local searches. The `ryftdec` stays ahead and translates complex search queries into several calls to the backend, like this:



In cluster mode, we use a set of `ryfthttp` search engines to access remote Ryft servers on the cluster, and `ryftmux` to multiplex all the results received:



`ryftprim` Search Engine

The `ryftprim` search engine uses the `ryftprim` command line tool to access the Ryft ONE server. Internally, the `ryftprim` tool uses `libryftone`, but to ensure thread-safe limitations of `libryftone` a separate process is spawned for each search call. Implementation sends the corresponding search command via the `ryftprim` tool, and then parses the generated index and data files.

`ryftprim` Options

The `ryftprim` search engine supports the following options (these can be customized via search configuration file):

Option	Description
instance-name	The name of the search engine instance. Used to distinguish different instances.
ryftprim-exec	The path to the `ryftprim` tool. Default is "/usr/bin/ryftprim."
ryftprim-legacy	Used to obtain machine-readable statistics. Default is true.
ryftone-mount	The main volume on the Ryft ONE. Default is "/ryftone."

Option	Description
open-poll	The open file poll timeout. The search engine continually attempts to open the index or data file within this timeout period. Default is 50ms.
read-poll	The read file poll timeout. The search engine will continually attempt to read the index or data file within this timeout period. Default is 50ms.
read-limit	The limit of read attempts. Default is 100. After the 100 fail read attempts, the search engine stops reading and returns an error.
keep-files	Keeps the intermediate data and index files in order to implement the `--keep` option on the server.
index-host	The node name of the cluster. The search engine marks all found record indexes with the name of the cluster node to distinguish where the record was found.

NOTE: The working directory for the `ryftprim` search engine is "ryftone-mount/$instance-name."

`ryftone` Search Engine

The `ryftone` search engine uses `libryftone` library to access Ryft hardware (See Ryft Open API)

Implementation is very similar to `ryftprim` search engine. It also sends corresponding search command to `libryftonelibrary` and then parses generated index and data files.

For future implementation.

`ryftone` Options

The `ryftone` search engine supports the same options as `ryftprim`, above.

`ryfthttp` Search Engine

The `ryfthttp` search engine uses another `ryft-server` instance to access Ryft hardware. This search engine is used in cluster mode to forward search queries to remote Ryft boxes.

`ryfthttp` Options

The `ryfthttp` search engine supports the following options:

Option	Description
server-url	Remote `ryft-server` address including host name and port. Default is "http://localhost:8765".

Option	Description
local-only	Flag to use local search on remote Ryft box. Related to `local=` server's query parameter
skip-stat	Flag to skip statistics. Related to `stats=` server's query parameter
index-host	Cluster node name. Search engine marks all found record indexes with cluster node's name. This feature lets you distinguish where the record come from. Usually, this option is not applied because all found indexes should be already marked by the local search engine (`ryftprim` or `ryftone`).

Usually these options are automatically filled by the `ryft-server` when the cluster configuration is built. It is also possible to customize these via search configuration file.

`ryftmux` Search Engine

The `ryftmux` search engine multiplexes results from several search engines, such as from one `ryftprim` and a few `ryfthttp` from cluster's nodes. This search engine is configured and used internally by the `ryft-server`.

There are no options for `ryftmux`.

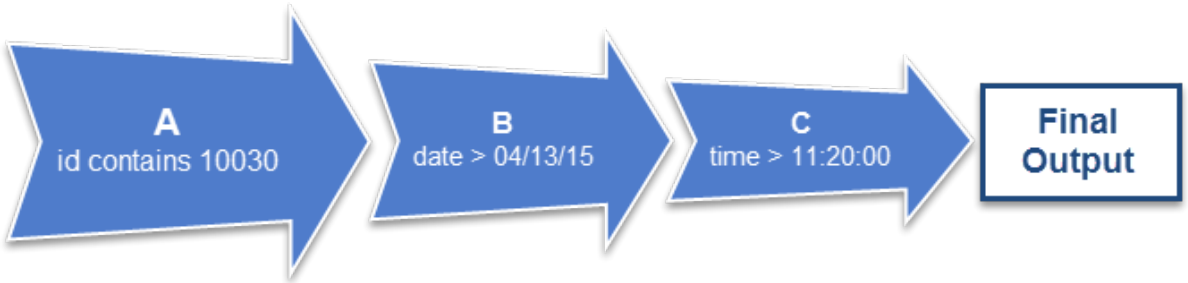
`ryftdec` Search Engine

The `ryftdec` search engine behaves similarly to a filter. It uses the `ryftprim` search engine instance in the background for query execution. The main purpose is to decompose complex search query into several simple sub-queries. These simple sub-queries are forwarded to the backend and the results are properly combined.

For example, let's process this complex search query with three different sub-queries:

```
(RECORD.id CONTAINS "10030") AND (RECORD.date CONTAINS DATE(MM/DD/YYYY > 04/13/2015))
AND (RECORD.date CONTAINS TIME(HH:MM:SS > 11:20:00))
```

The `ryftdec` search engine decomposes the complex expression into the following tree:



The expression A is called first, as part of a normal search. The results of A are used as input for the B date search sub-query. The results of B are then used as input for the C time search sub-query.

NOTE: For structured search, it's critical to keep the temporary file extension the same as the input file. For example, if the input file ends with “*.pcrime” then the temporary file must also have the same “*.pcrime” extension. Otherwise, the Ryft hardware won't use the corresponding RDF scheme.

2

2: Running Ryft REST Server

Run Server

You can run the `ryft-server` on a single instance, or run multiple instances on different ports by running the appropriate command:

- Single instance:

```
/usr/bin$ ./ryft-server
```

- Multiple instances on different ports. This command runs another server instance on port 9000 in debug mode.

```
/usr/bin$ ./ryft=server 0.0.0.0:9000 --debug
# or
/usr/bin$ ./ryft-server -l=:9000 --debug
```

It is possible to run multiple server instances on the same machine.

Use `ryft-server --help` to display the list of all supported arguments.

The ryft-server Daemon

The normal operation is to run the `ryft-server` as a daemon. The `ryft-server` daemon automatically starts on boot with default arguments, using port 8765 as the listening port.

The service can be manually stopped with this command:

```
$ sudo service ryft-server-d stop
```

Manually start the service using this command:

```
$ sudo service ryft-server-d start
```

Set Arguments

You can pass arguments to the `ryft-server` daemon in the "/etc/ryft-server.conf" file and restarting `ryft-server-d` service.

List of Arguments

The list of available arguments can be easily obtained by executing `ryft-server --help`.

```
~/rest$ ./ryft-server --help
usage: ryft-server-test [<flags>] [<address>]

Flags:
      --help                     Show context-sensitive help (also try --help-long and
--help-man).
      --config=CONFIG            Server configuration in YML format.
  -k, --keep                     Keep search results temporary files.
  -d, --debug                    Run http server in debug mode.
  -a, --auth=AUTH                Authentication type: none, file, ldap.
      --users-file=USERS-FILE    File with user credentials. Required for --auth=file.
      --ldap-server=LDAP-SERVER  LDAP Server address:port. Required for --auth=ldap.
      --ldap-user=LDAP-USER      LDAP username for binding. Required for --auth=ldap.
      --ldap-pass=LDAP-PASS      LDAP password for binding. Required for --auth=ldap.
      --ldap-query="(&(uid=%s))"
                                 LDAP user lookup query. Defauls is '(&(uid=%s))'.
Required for --auth=ldap.
      --ldap-basedn=LDAP-BASEDN  LDAP BaseDN for lookups.'. Required for --auth=ldap.
  -t, --tls                      Enable TLS/SSL. Default 'false'.
      --tls-crt=TLS-CRT          Certificate file. Required for --tls=true.
      --tls-key=TLS-KEY          Key-file. Required for --tls=true.
      --tls-address=0.0.0.0:8766
                                 Address:port to listen on HTTPS. Default is
0.0.0.0:8766

Args:
  [<address>]  Address:port to listen on. Default is 0.0.0.0:8765.
```

Keeping Search Results

The `ryft-server` uses a dedicated directory on the "/ryftone" partition to store temporary results and index files that `ryft-server` creates. The instance directory name is auto-generated using the port number, "/ryftone/.rest-$PORT/" and can be customized via the search configuration file (see below).

By default, the `ryft-server` removes all search results from the "/ryftone/.rest-$PORT/" directory. You can override this function and keep the results by using the `--keep` flag:

```
$ ./ryft-server --keep
```

All temporary result files will be kept under the server's instance directory. This feature is useful for troubleshooting as the files are available for use.

Configuration File

The `ryft-server` supports additional configuration files. This YAML configuration file can be customized with the `--config` command line option, like this:

```
$ ./ryft-server --config=$path_to_yaml_config_file
```

Additionally, if the configuration file is located at "/etc/ryft-server.conf" then it is automatically used by the `init` script when the service starts. The default configuration is provided by the Debian package.

A sample "ryft-server.conf" file is shown on the following page.

```
ryftuser@R01-0311:/etc$ cat ryft-server.conf
### this configuration file contains ryft-server options.
### most of the options may be overriden by corresponding
### command line option (noted in parenthesis).

### main search engine and its options
search-backend: ryftprim
backend-options:
  ryftprim-legacy: true
  ryftprim-exec: /usr/bin/ryftprim
  ryftone-mount: /ryftone
  open-poll: 100ms
  read-poll: 100ms
  read-limit: 100

### start listening on this address and port (--address)
# address: 0.0.0.0:8765

### HTTPS support (--tls, --tls-address, --tls-cert, --tls-key)
tls:
  enabled: false
  address: 0.0.0.0:8766
  cert-file: "<certificate file name>"
  key-file: "<key file name>"

### authentication type: none, file, ldap (--auth)
auth-type: none

### JWT authentication (--jwt-alg, --jwt-secret, --jwt-lifetime)
### secret may be simple string or file reference, for example "@~/.ssh/id_rsa"
auth-jwt:
  algorithm: HS256
  secret: "<secret key>"
  lifetime: 1h

### file based authentication (--users-file)
auth-file:
  users-file: /etc/ryft-users.yaml

### LDAP based authentication (--ldap-server, --ldap-user, --ldap-pass, --ldap-query,
--ldap-basedn)
auth-ldap:
  server: ldap.forumsys.com:389
  username: "read-only-admin,dc=ryft,dc=one"
  password: "password"
  query: "(&(cn=%s))"
  basedn: "dc=ryft,dc=one"
#  insecure-skip-tls: true
#  insecure-skip-verify: true


### run server in local mode (no cluster, no consul, no load balancing) (--local-only)
local-only: true

### run server in debug mode - a lot of log messages (--debug)
# debug-mode: true

### keep intermediate INDEX and DATA files, used for debugging (--keep)
# keep-results: true

### busyness tolerance (--busyness-tolerance)
# busyness-tolerance: 1
```

```
### number of possible boolean operators per expression type
booleans-per-expression:
  es:   1
  fhs:  1
  feds: 1
  ns:   0
  ds:   1
  ts:   1
  rs:   0
```

Search Configuration

Using a configuration file, it is possible to change the main search engine and its options. This is the file format:

```
### main search engine and its options
search-backend: <search engine>
backend-options:
  <search engine options>
```

The `search-backend` is the search engine name and can be one of the following:

- `ryftprim` uses the `ryftprim` command line tool to access Ryft server (used by default).
- `ryftone` uses the `libryftone` library to access Ryft server.
- `ryfthttp` uses another `ryft-server` instance to access Ryft server.

The `backend-options` is the search engine specific option. For example, `ryftprim` engine supports the following options and would look like this:

```
search-backend: ryftprim
backend-options:
  instance-name: .ryft/8765   # server instance name (.rest-$PORT by default)
  ryftprim-exec: ryftprim     # ryftprim tool path (/usr/bin/ryftprim by default)
  ryftone-mount: /ryftone     # ryftone volume (/ryftone by default)
                              # server instance directory will be:
                              #    $ryftone-mount/$instance-name
```

Server Configuration

This configuration file also contains most of the command line options that also can be customized. Note that you can override these options by the command line. For example, if your configuration file contains this:

```
### start listening to this address and port (--address)
address: 0.0.0.0:8000
```

But the server starts as this:

```
ryft-server -config=/etc/ryft-server.conf -address=0.0.0.0:9000
```

Then the actual option for the address will be 0.0.0.0:9000, since it goes last.

Using the `address` option, it's possible to customize the server's listen address. Its equivalent to the `--address` command line option. By default, "0.0.0.0:8765" is used.

There are also a few more options:

```
local-only: false
debug-mode: false
keep-results: false
busyness-tolerance: 0
http-timeout: 1h
```

Option	Description
local-only	Use to run `ryft-server` outside cluster. No consult dependency, no load balancing enabled. It's equivalent to using the `--local-only` command line option.
debug-mode	Us to enable extensive logging. It's equivalent to using the `--debug` command line option.
keep-results	Use to keep intermediate INDEX and DATA files for debugging. It's equivalent to using the `--keep` command line option.
busyness-tolerance	Use in cluster mode to customize node grouping algorithm. See Ryft Cluster documentation for more details. It's equivalent to using the `--busyness-tolerance` command line option.
http-timeout	Use as read request/write response timeout for HTTP/HTTPS connections. By default, set to `1h` (one hour).

TLS Server Configuration

Configure the Transport Layer Security (TLS) protocol server to run with HTTPS enabled . Use the `--tls`, `--tls-address`, `--tls-cert`, and `--tls-key` command line options or just the corresponding `tls` section in the configuration file:

```
### HTTPS support (--tls, --tls-address, --tls-cert, --tls-key)
tls:
  enabled: false
  address: 0.0.0.0:8766
  cert-file: "<certificate file name>"
  key-file: "<key file name>"
```

The HTTPS can be enabled or disabled using the boolean `enabled` flag. The listen port address can be customized using the `address` option. Note that the port number should be different from the normal address.

Two files should be provided to enable HTTPS:

- Certificate file `cert-file`
- Corresponding certificate key `key-file`.

Authentication Server Configuration

There are a few sections related to authentication.

```
### authentication type: none, file, ldap (--auth)
auth-type: none
```

The `auth-type` is used to select authentication provider. It can be one of the following options:

- `auth-type: none` – disables authentication.
- `auth-type: file` – enables authentication
- `auth-type: ldap` – enables authentication

JWT Authentication – `auth-jwt`

If JSON Web Token (JWT) authentication is enabled, then include the `auth-jwt` section:

```
### JWT authentication (--jwt-alg, --jwt-secret, --jwt-lifetime)
### secret may be simple string or file reference, for example "@~/.ssh/id_rsa"
auth-jwt:
  algorithm: HS256
  secret: "<secret key>"
  lifetime: 2h
```

- Customize the sign in by using the algorithm option. In this example, it uses HS256.
- The `secret` can be simple string or file reference:
  - `secret: "my super secret key"` or
  - `secret: "@~/.ssh/id_rsa"` to use `~/.ssh/id_rsa` file content as a secret.
- JWT token lifetime can be customized via `lifetime` option. By default, it's `lifetime: 1h`.

File Based Authentication - `auth-file`

If simple file is used as an authentication provider `auth-type: file`, provide the user credentials like this:

```
### file based authentication (--users-file)
auth-file:
  users-file: /etc/ryft-users.yaml
```

The file formats are described in the Authentication section below.

LDAP Based Authentication - `auth-ldap`

If Lightweight Directory Access Protocol (LDAP) is used as an authentication provider `auth-type: ldap`, provide the LDAP credentials:

```
### LDAP based authentication (--ldap-server, --ldap-user, --ldap-pass, --ldap-query,
--ldap-basedn)
auth-ldap:
  server: ldap.forumsys.com:389
  username: "read-only-admin,dc=ryft,dc=one"
  password: "<password>"
  query: "(&(cn=%s))"
  basedn: "dc=ryft,dc=one"
#  insecure-skip-tls: true
#  insecure-skip-verify: true
```

- Use `server` to customize the LDAP server address.
- Use `username` and `password` to customize the read-only user which is used to send search request to the LDAP service.
- Use `query` and `basedn` to specify attribute name which is used to search and base DN.

There are also a few options related to security. By default, `ryft-server` tries to connect LDAP using TLS.

- To disable TLS, set `insecure-skip-tls: true`.
- To disable certificate verification (may be useful if LDAP uses self-signed certificate), set `insecure-skip-verify: true`.

NOTE: Ryft does not recommend defining these `insecure-*` options in your production environment.

Authentication

The following types of authentication are supported (hyperlinks take you to other websites):

- Basic - Basic Access Authentication
- JWT – JSON Web Tokens

To verify user credentials, the LDAP service or simple file may be used.

The following endpoints are protected:

- /search
- /count
- /files

If authentication is enabled, the `ryft-server` checks for Authorization HTTP header.

If the Authorization header contains the keyword “Basic,” the basic authentication is used. The `ryft-server` extracts username and password from the header and checks that the user is authorized to access the requested resources.

Otherwise, if the Authorization header contains keyword “Bearer,” the JWT is used. The `ryft-server` extracts JWT token from the header and uses it.

There are two special endpoints for JWT authentication:

- `/login` - used to get JWT token
- `/token/refresh` - used to refresh existing token

JWT Login

The `/login` endpoint expects `{"username":"login", "password":"password"}` JSON structure as an input. If the credentials are valid, the JWT token is provided, as a result.

Here’s an example:

```
curl -d '{"username":"admin","password":"admin"}' "localhost:8765/login"
```

It returns the following:

```
{"expire":"2016-07-11T08:13:09-04:00",
"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE0NjgyMzkxODksImlkIjoiYWRtaW4i
LCJvcmlnX2lhdCI6MTQ2ODIzNTU4OX0.X_sO1pimiDQ9XGg37PzTYIB9ohu4DJM8VG9lgqd4sqg"}
```

Having this token, it's now possible to send authorized requests, like this one:

```
TOKEN=`curl -d '{"username":"testuser","password":"testpsswd"}' "localhost:8765/login"
| jq -r .token`
curl -H "Authorization: Bearer $TOKEN"
http://localhost:8765/search?query=Joe&files=*.txt
```

JWT Options

To pass JWT secret to the server, use the configuration file or `--jwt-secret` command line option:

```
ryft-server --jwt-secret=my-secret-key
ryft-server --jwt-secret=@my-secret-file
ryft-server --jwt-secret=hex:6D792D7365637265742D6B6579
ryft-server --jwt-secret=base64:bXktc2VjcmV0LWtleQ==
```

By default, `HS256` signing algorithm is used. You can override it by using the `--jwt-alg` command line option, like this: `--jwt-alg=RS256`.

The default token lifetime is 1 hour. To change it, use `--jwt-lifetime` command line option. The overall token refresh timeout is 10 lifetimes.

LDAP

Most LDAP customization can be done via the configuration file. Some command line options are also available. (Check `ryft-server --help` output for more information.)

- Provide the LDAP server address. See `--ldap-server` command line option.
- Provide special read-only account for search requests. Customize the account credentials using the `--ldap-user` and `--ldap-pass` command line options.

The `--ldap-query` and `--ldap-basedn` are used to finish the LDAP search request. Query format is used to select appropriate RDN, for example `(&(cn=%s))`.

There are a few security related options (currently in configuration file only) that could be used to disable TLS or to disable TLS certificate verification. Please check corresponding auth-ldap section of the configuration file.

Simple Text File

Simple text file may be used as a list of user credentials. Here are two examples in YAML and JSON format.

YAML format:

```
- username: "admin"
  password: "admin"
  home: "/"
- username: "test"
  password: "test"
  home: "/test"
  cluster-tag: "test"
- username: "foo"
  password: "foo"
  home: "/foo"
  cluster-tag: "foo"
```

JSON format:

```
[
  {"username":"admin", "password":"admin", "home":"/"},
  {"username":"test", "password":"test", "home":"/test", "cluster-tag":"test"},
  {"username":"foo", "password":"foo", "home":"/foo", "cluster-tag":"foo"}
]
```

The home is a directory inside the "/ryftone" mount point. It is used to separate data of various users. A user is only authorized to access its home directory.

The `cluster-tag` is used for partitioning. If a user has custom partitioning rules, they are located under the `{cluster-tag}/partitions` KV prefix.

To run the server, use the following command line:

```
ryft-server --auth=file --users-file "ryft-users.yaml"
```

Cluster Mode

All cluster nodes should have the same list of users (or the same LDAP configured) and the same secret key. It's important to be able to pass authentication tokens between cluster nodes. Ryft server uses Authorization HTTP header "as is" to redirect search requests.

Moreover, each user should have the same home directory on each cluster node.

Log File

The `ryft-server-d` service log file, "ryft-server-d-start.log," is located in the ryftuser home directory.

To view logs in real-time:

```
$ tail -f ~/ryft-server-d-start.log
```

3

3: REST API

The `ryft-server` supports these REST endpoints:

- /search
- /count
- /files
- /version

NOTE: All examples in this section assume that "ryftone-777" is the `ryft-server` host name.

The main API endpoints are `/search` and `/count`. Both have similar parameters. However, the `/count` endpoint does not transfer all found data. Instead, it prints the number of matches and associated performance numbers. The minimum required parameters are search query and the set of files to search.

If authentication is enabled, there are also a few endpoints related to JWT.

Search Endpoint - `/search`

The `GET /search` endpoint is used to search data on Ryft servers. Note, this endpoint is protected and user should provide valid credentials. See Authentication for more details.

Here is the list of supported query parameters. Detailed description for each follows:

Parameter	Type	Description
`query`	string	Required. The search expression.
`files`	string	Required. The set of files to search. Wildcards are supported.
`mode`	string	The search mode. If no search mode is specified, fuzzy hamming search is used, by default, for simple queries.

Parameter	Type	Description
`surrounding`	uint16	The data surrounding width. Number of characters to return before and after the search string. Used with RAW text search. Default is `surrounding=0`.
`fuzziness`	uint8	The fuzziness distance. Measured as the maximum Hamming distance for “fhs” (fuzzy hamming search) mode, or edit distance for “feds” (fuzzy edit distance search) mode. Default is `fuzziness=0`.
`format`	string	The structured search input data format. Used with structured search.
`cs`	boolean	The case sensitive flag. Default is `cs=false`.
`fields`	string	The set of fields to get. Comma-separated list of requested fields. Use with structured search to minimize output.
`data`	string	The name of data file to keep. File will be overwritten.
`index`	string	The name of index file to keep. File will be overwritten.
`nodes`	integer	The number of processing nodes to use (1-4). If omitted, all available nodes are used.
`local`	boolean	The local/cluster search flag. Default is `local=false`, cluster mode.
`stats`	boolean	The statistics flag. Include search statistics in results. Default is `stats=false`.
`limit`	int	Limit the total number of records reported.
`stream`	boolean	Internal. The stream output format flag.
`ep`	boolean	Internal. The error prefix flag. Default is `ep=false`.

Search Parameter: `query`

This parameter enables you to use one or more subqueries, connected using logical operators. It can be used for both unstructured and structured searches.

To execute an unstructured text search for "The Batman" use this search expression:

```
query=(RAW_TEXT CONTAINS "The Batman")
```

To execute a structured search, use this expression to define the field to search for the same text.

```
query=(RECORD.AlterEgo CONTAINS "The Batman")
```

Depending on which search mode you use, the exact search query format may differ.

See Ryft Open API or General Search Syntax section for more details on search expressions.

Simple/Plain Queries

Use `ryft-server` to execute simple, plain queries without any key words. From our previous example, the statement `query=Batman` will automatically convert to `query=(RAW_TEXT CONTAINS "Batman")`. (In the background, the query is translated to be: `query=(RAW_TEXT CONTAINS "\x42\x61\x74\x6d\x61\x6e")`, because `ryft-server` uses hex encoding to avoid any possible escaping problems).

NOTE: The statement `query=Batman` only works for unstructured text search. It is not supported for structured search.

Complex Queries

You can also execute complex queries that contain several different search expressions at one time.

Here is an example that contains two search expressions: text search and date search:

```
(RECORD.id CONTAINS "100") AND (RECORD.date CONTAINS DATE(MM/DD/YYYY > 04/15/2015))
```

The `ryft-server` command splits this expression into two separate queries. It then calls the Ryft hardware two times so that the results of the first call are used as the input for the second call, like this:

```
(RECORD.id CONTAINS "100")
(RECORD.date CONTAINS DATE(MM/DD/YYYY > 04/15/2015))
```

Multiple `AND` and `OR` operators are supported by the `ryft-server` within complex search queries. An expression tree is built and each node is passed to the Ryft hardware, and then the results are properly combined.

NOTE: If the search query contains two or more expressions of the same type (text, date, time, numeric), then that query will not be split into subqueries because the Ryft hardware supports those type of queries directly.

It is also possible to use advanced text search queries to customize some parameters within search expression. Here is an example:

```
(RAW_TEXT CONTAINS FHS("555",CS=true,DIST=1,WIDTH=2)) AND (RAW_TEXT CONTAINS
FEDS("777",CS=true,DIST=1,WIDTH=4))
```

The `ryft-server` splits this expression into two Ryft calls:

```
(RAW_TEXT CONTAINS "555") with fhs search mode, fuzziness=1 and surrounding=2
(RAW_TEXT CONTAINS "777") with feds search mode, fuzziness=1 and surrounding=4
```

This advanced search query syntax overrides the following global parameters:

- search type: `FHS` or `FEDS` (exact search is used if fuzziness is zero)
- case sensitivity: `CS=`
- fuzziness distance: `DIST=`
- surrounding width: `WIDTH=`

If nothing is provided, the global options are used by default.

Any option may be omitted, like this example:

```
(RAW_TEXT CONTAINS FHS("555")) AND (RAW_TEXT CONTAINS FEDS("777",CS=false)).
```

Search Parameter: `files`

Use the `files` parameter to specify search input file(s). At least one file should be provided. Multiple files can be provided as a list or wildcard, like this:

- List: `files=1.txt&files=2.txt&files=3.txt`
- Wildcard: `files=*txt`

Search Parameter: `mode`

There are many search modes available with the `ryft-server`:

- `es` for exact search
- `fhs` for fuzzy hamming search (default, if no mode is specified)
- `feds` for fuzzy edit distance search
- `ds` for date search
- `ts` for time search
- `ns` for numeric search
- `ipv4` for IPv4 search
- `ipv6` for IPv6 search

NOTE: If no search mode is specified, then by default fuzzy hamming search is used for simple queries.

It is also possible to automatically detect search modes. If the search query contains the keyword "DATE" then the `DATE` search will be used. It's the same when the keyword "TIME" is used for `TIME` search, and "NUMBER" or "CURRENCY" keywords for `NUMERIC` search.

In case of complex search queries, the specified mode is used for text or structured search, only. Date, time and numeric search modes are automatically detected by their corresponding keywords.

NOTE: The fuzzy edit distance search mode removes duplicates, by default (`-r` option of `ryftprim`).

Search Parameter: `surrounding`

Use the `surrounding` parameter with raw text search to specify the number of characters to return both before and after the match. You can specify up to a maximum of 262,144 bytes before and after the match. For anything other than raw text, this parameter is ignored.

By default, `surrounding=0`.

Search Parameter: `fuzziness`

The fuzziness of the search means different things, depending on the type of fuzzy search you perform. You can specify up to a maximum fuzziness of 255 when using either of these fuzzy search functions:

- Fuzzy Hamming Search (`fhs`): Fuzziness is measured as the maximum Hamming distance allowed in order to declare a match.
- Fuzzy Edit Distance Search (`feds`): Fuzziness is measured as the number of insertions, deletions or replacements required to declare a match.

By default, `fuzziness=0`.

Search Parameter: `format`

Use `format` to specify the input data format for structured search.

By default, structured search uses `format=raw` format. That means that found data are reported as base-64 encoded raw bytes.

There are three other options: `format=xml`, `format=json`, and `format=utf8`.

XML Format

If the input file set contains XML data, the found records could be decoded by using `format=xml` query parameter. Records will then be translated from XML to JSON.

JSON Format

If the input file set contains JSON data, use the `format=json` query parameter to decode data to JSON

UTF8 Format

Use `format=utf8` to get human readable data (instead of base-64 encoded bytes). This parameter asks `ryft-server` to interpret found bytes as UTF-8 string:

```
curl "http://ryftone-777:8765/search?query=Joe&files=*.txt&format=utf8"
```

A `format=raw` - base-64 encoded raw bytes sample may look like this:

```
{
  "data": "LDMxMC01NTUtMzQyNQ==",
  "_index": {}
}
```

That same sample displayed as `format=utf8` would look like this:

```
{
  "data": ",310-555-3425",
```

```
  "_index": {}
}
```

Search Parameter: `cs`

Use the `cs` parameter as a flag to specify the search text case-sensitivity. By default, `cs=false`.

For example, if the search is case-sensitive (`cs=true`), then searching for the string "John" will not find any occurrences of "JOHN". If the same search is done with `cs=false`, then case is ignored entirely and all possible capitalizations of the text will be found (e.g. "jOhn" or "JOHn").

Search Parameter: `fields`

The comma-separated list of fields to use for structured search. If omitted, all fields are used.

This parameter is used to minimize structured search output or to get just subset of fields. For example, to get identifier and date from a `*.pcrime` file, pass `format=xml&fields=ID,Date`.

The same is true for JSON data: `format=json&fields=Name,AlterEgo`.

Search Parameters: `data` and `index`

By default, all search results are deleted from the Ryft server once they are delivered to the user. The `data` and `index` parameters enable you to preserve the results, giving you the ability to use the results as the input for a second, subsequent, search.

WARNING: The data or index files are overridden.

Data Parameter

Using the `data=output.dat` parameter creates the search results file “output.dat” on the Ryft server under “/ryftone/output.dat.” It is then possible to use “output.dat” as the input file for the subsequent search call using `files=output.dat`.

NOTE: It is important to use consistent file extensions for the structured search in order to let Ryft use the appropriate RDF scheme.

Index Parameter

Using the second parameter `index=index.txt` keeps the search index file under the “/ryftone” directory.

NOTE: As noted in the Ryft API guide, an index file should always have `.txt` extension.

Search Parameter: `nodes`

Use the `nodes` parameter to specify the number of Ryft processing nodes that the algorithm should use. A minimally configured Ryft ONE ships from the factory with one processing node (`nodes=1`), and a maximally configured Ryft ONE ships with four processing nodes (`nodes=4`).

If you omit the parameter, the maximum of available nodes is used.

Search Parameter: `local`

The local/cluster flag. By default, `ryft-server` uses cluster mode (`local=false`) which means that the `ryft-server` asks all appropriate nodes in the cluster and then combines the results.

To execute a search on single node use `local=true`.

Search Parameter: `stats`

By default, statistics is not reported (`stats=false`). To check total number of matches and performance numbers, use `stats=true`.

Search Parameter: `limit`

The `limit` parameter is used to limit the total number of records reported. By default, there is no limit, or when you specify `limit=0`.

Search Parameter: `stream`

The `ryft-server` reports results in several formats. By default, the simple JSON object with "results" array and "stats" object is reported. That format is used by the Ryft Web-UI:

```
{
  "results": [
    {
      "Date": "04/15/2015 11:59:00 PM",
      "ID": "10034183",
      "_index": {}
    },
    {
      "Date": "04/15/2015 11:59:00 PM",
      "ID": "10034184",
      "_index": {}
    }
  ],
  "stats": {
    "matches": 2,
    "totalBytes": 6902619,
    "duration": 415,
    "dataRate": 15.862290255994683,
    "fabricDataRate": 15.86229
  }
}
```

But this format is not efficient for cluster nodes communication. We cannot decode a JSON object until all of the data is received. Instead, use the `stream` parameter - a sequence of JSON "tag-object" pairs that enables `ryft-server` to decode input data, on the fly:

```
"rec"
{
  "Date": "04/15/2015 11:59:00 PM",
  "ID": "10034183",
  "_index": {}
}

"rec"
{
  "Date": "04/15/2015 11:59:00 PM",
```

```
  "ID": "10034184",
  "_index": {}
}

"stat"
{
  "matches": 2,
  "totalBytes": 6902619,
  "duration": 1456,
  "dataRate": 4.521188500163319,
  "fabricDataRate": 4.521189
}

"end"
```

Search Parameter: ep

The ep parameter is the helper "error prefix" flag for cluster mode. The default is ep=false.

To let the user know which cluster node reported the error, ryft-server adds the node's hostname to each error message.

Search Examples

Example 1: Simple Example

This is an example of a simple search request to find “Joe” in all files that end in “*.txt”:

```
/search?query=Joe&files=*.txt
```

Example 2: Single Node Example

By default, cluster mode is used for all searches. To execute a search on a single node use the local query parameter:

```
/search?query=Joe&files=*.txt&local=true
```

Example 3: Unstructured Search Example

This is an example of an unstructured search request to find “10” in the file “passengers.txt.” The output will contain 12 bytes of data surrounding the search value, and also output stats on the results. The request will be run on only on the local server vs. the cluster.

```
/search?query=10&files=passengers.txt&surrounding=12&fuzziness=0&stats=true&local=true
```

Example 4: Structured Example

This is an example of a structured search request to find “1003100” in the field “id” within all files that end in “*.pcrime” located in the “/” directory. It must be an exact match as fuzziness is set to 0, the format will be XML, and the output will contain the fields ID and Date. The output will also contain stats, and the query will run on the local server vs. the cluster.

```
/search?query=(RECORD.id CONTAINS
"1003100")&files=/*.pcrime&fuzziness=0&format=xml&fields=ID,Date&stats=true&local=true
```

Example 5: Fuzzy Edit Distance Example

The following command captures 5 bytes of data surrounding the search value, and executes a fuzzy edit distance search (`fuzziness=2`) instead of fuzzy hamming search, which is used by default:

```
curl "http://ryftone-
777:8765/search?query=Joe&files=*.txt&mode=feds&surrounding=5&fuzziness=2"
# - or -
curl --get --data-urlencode 'query=(RAW_TEXT CONTAINS "Joe")' \
  "http://ryftone-777:8765/search?files=*.txt&mode=feds&surrounding=5&fuzziness=2"
```

Count Endpoint - `/count`

The `GET /count` endpoint is also used to search data on Ryft boxes. It prints the number of matches and associated performance numbers. It does not transfer all found data.

NOTE: This endpoint is protected and user is prompted to provide valid credentials. See authentication for more details.

The parameters are:

Parameter	Type	Description
query	String	Required. The search expression.
files	String	Required. The set of files to search.
mode	String	The search mode.
surrounding	Uint16	The data surrounding width.
fuzziness	Uint8	The fuzziness distance.
cs	Boolean	Case sensitive flag.
data	String	Name of the data file to keep.
index	String	Name of index file to keep.
nodes	Integer	Number of processing nodes. Default is maximum number available on the Ryft server.
local	Boolean	True or False. Local/cluster search flag. By default, set to False/Cluster.

Example 1

Here is an example of counting the number of matches where the letter “a” or “b” is contained in all data files that end with *.pcrime, on the local server:

```
/count?query=(RECORD CONTAINS “a”) OR (RECORD CONTAINS “b”)&files=*.pcrime&local=true
```

Example 2

Here is an example of counting the number of matches where record contains “1003” in “id” and the date is > 04/14/2015 and the time is > 08:30:00 on the “mychicago.pcrime” file, on the local server:

```
/count?query=(RECORD.id CONTAINS "1003") AND (RECORD.date CONTAINS DATE(MM/DD/YYYY >
04/14/2015)) AND (RECORD.date CONTAINS TIME(HH:MM:SS >
08:30:00))&files=mychicago.pcrime&local=true
```

Example 3

Here is an example that uses AND and OR so that either the first criteria (record contains “1003” and date <= 04/12/2015) or the second criteria (time search) is met.

```
/count?query=(RECORD.id CONTAINS "1003") AND ((RECORD.date CONTAINS DATE(MM/DD/YYYY <=
04/12/2015)) OR (RECORD.date CONTAINS TIME(HH:MM:SS >
11:15:00)))&files=mychicago.pcrime&local=true
```

Files Endpoint - `/files`

The `GET /files` endpoint is used to get a list of the directories, subdirectories and files for the Ryft server node or cluster.

NOTE: This endpoint is protected and user is prompted to provide valid credentials. See authentication for more details.

The parameters are:

Parameter	Type	Description
`dir`	String	Directory from which to get content. By default, it is the root “/ryftone” directory. Directory name should be relative to the Ryft volume, so using `dir=/test` means report the contents of /ryftone/test directory on the Ryft server.
`local`	Boolean	The local/cluster flag. True or false. By default, it is false, which means it is cluster.

For example, this command means to print the contents of the “/”ryftone directory on the local server:

```
/files?dir=/&local=true
```

The resulting output may look like this:

```
{
  "dir": "/",
  "files": [
    "chicago.pcrime",
    "passengers.txt"
  ],
  "folders": [
    "demo",
    "regression",
    "test"
  ]
}
```

Current Server Version - `/version`

The `GET /version` endpoint is used to verify the current build version of `ryft-server`. This request prints the current server version and corresponding Git hash number. This information is extremely useful for bug reporting. There are no parameters.

```
curl http://ryftone-777:8765/version
```

The resulting output may look like this:

```
{
  "git-hash": "35c358378f7c214069333004d01841f9066b8f15",
  "version": "1.2.3"
}
```

Logical Operator Priority: `AND` / `OR` / `NOR`

It is important to understand the priority of the logical operators for queries which combine AND / OR operations. When used together, first priority is given to the `AND` operator, then the `OR` operator, which means that **a OR b AND c** is equivalent to **a OR (b AND c)**.

General Search Syntax

A match criteria `query` parameter is used to specify how the search should be performed. Search criteria are made up of one or more relational expressions, connected using logical operations. The Ryft Open API defines query language grammar as consisting of a relational expression that takes the following form:

```
(input_specifier relational_operator expression)
```

The `input_specifier` specifies how the input data is arranged. The possible values are:

- `RAW_TEXT` - The input is a sequence of raw bytes with no implicit formatting or grouping.
- `RECORD` - The input is a series of records. Search all records.
- `RECORD.<field_name>` - The input is a series of records. Search only the field called `<field_name>` in each record. **Note**: for JSON input records, multiple field names can be specified with '.' separators between them to specify a field hierarchy, or with '[]' separators to specify array hierarchy.

The `relational_operator` specifies how the input relates to the expression. The possible values are:

- `EQUALS` - The input must match the expression exactly for an exact search, or within the specified Hamming distance for a fuzzy search, with no additional leading or trailing data. This operator only has meaning for record- and field-based searches. If used with a raw text operation, an error is generated; use `CONTAINS` instead.
- `NOT_EQUALS` - The input must be anything other than expression. This operator has meaning only for record- and field-based searches. If used with a raw text operation, an error is generated.
- `CONTAINS` - The input must contain expression, and may contain additional leading or trailing data.

- `NOT_CONTAINS` – The input must not contain expression. This operator has meaning only for record- and field-based searches. If used with a raw text operation, an error is generated.

The `expression` specifies the expression to be matched. The possible values are:

- Quoted string - Any valid C language string, including backslash-escaped characters. For example, look at the string "`match this text\n`. This can also include escaped hexadecimal characters, such as:
  - `match this text\x0A`, or
  - `\x48\x69\x20\x54\x68\x65\x72\x65\x21\x0A\x00`.
- If a backslash needs to be placed in the quoted string for search query purposes, use the double backsplash escape sequence "`\\`" so that it is escaped properly.
- Wildcard - A `?` character is used to denote that any single character will match. It can be inserted at any point(s) between quoted strings. For example, "`match th"?"s text\n`".
- Any combination of the above - For example, "`match\x20th"?"s text\x0A`", or "`match\x20with a wildcard right here"?"and a null at the end\x00`".

The `logical_operator` allows for complex collections of relational expressions. The possible values are `AND`, `OR` and `XOR`:

- The logical expression `(a AND b)` evaluates to true only if both the relational expression “a” evaluates to true and the relational expression “b” evaluates to true.
- The logical expression `(a OR b)` evaluates to true if either the relational expression “a” evaluates to true or the relational expression “b” evaluates to true.
- The logical expression `(a XOR b)` evaluates to true if either the relational expression “a” evaluates to true or the relational expression “b” evaluates to true, but not both.

Multiple relational expressions can be combined using the logical operators. For example:

```
(RECORD.city EQUALS "Rockville") AND (RECORD.state EQUALS "MD")
```

Parentheses can also be used to control the precedence of operations. For example:

```
((RECORD.city EQUALS "Rockville") OR (RECORD.city EQUALS "Gaithersburg")) AND
(RECORD.state EQUALS "MD")
```

Exact Search

The exact search operation will search `existing_data_set` using a match criterion. The match string can be up to 32 bytes in length. Only exact matches will be returned.

When the format of the input data is specified as raw text, the `surrounding` parameter specifies how many characters around the match will be returned as part of the matched data results. The `surrounding` width can be useful to assist a human analyst or a downstream machine learning tool to determine the contextual use of a specific matched term when searching unstructured raw text data.

Fuzzy Hamming Search

The Fuzzy Hamming search operation works similarly to exact search except that matches do not have to be exact. Instead, the fuzziness parameter allows the specification of a "close enough" value to

indicate how close the input must be to match the search criteria. The match string can be up to 32 bytes in length. A "close enough" match is specified as a Hamming distance.

The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. As provided to the fuzzy search operation, the Hamming distance specifies the maximum number of substitutions that are allowed in order to declare a match. In addition, similar to exact search, the surrounding mechanism can aid in downstream analysis of contextual use of the fuzzy match results against unstructured raw text data.

Fuzzy Hamming search algorithms are highly parallelizable, so they can run extremely quickly on accelerated hardware. Therefore, if fuzzy search algorithms can make use of fuzzy Hamming search, then they will typically perform at very high speed.

This means that if the search string is “united states” with a fuzziness of 2, then any match must first match the string length (13 characters) and up to 2 characters in the string may be different.

Fuzzy Edit Distance search

Fuzzy Edit Distance Search performs a search that does not require two strings to be of equal length to obtain a match. Instead of considering individual symbol differences, fuzzy edit distance search counts the minimum number of insertions, deletions and replacements required to transform one string into another. This can make it much more powerful than Fuzzy Hamming search for certain applications.

Let’s compare searching for the string “Michelle” using Fuzzy Hamming vs. Fuzzy Edit Distance, with an edit distance = 1.

String	Fuzzy Hamming, Edit Distance = 1	Fuzzy Edit Distance, Edit Distance = 1
Michelle	Yes. Exact match.	Yes. Exact match.
Mishelle	Yes. “c” changed to “s”.	Yes. “c” changed to “s”.
Mischelle	No. The string “chelle” does not appear in the same position as in the original search term “Michelle”. This makes it an edit distance of 6, which is greater than the specified distance.	Yes. Can insert the single character “s”.
Michele	No. Deleting one “l” shortens the string by one character, and the match string must be of equal length.	Yes. One “l” deleted.
Mischele	No. Although of equal length, more than one change is required to match.	No. Requires 2 changes: add an “s” and remove an “l”. **NOTE**: If the edit distance = 2, then it would match since the calculated edit distance between the 2 strings is less than or equal to the desired edit distance (2).

Fuzzy edit distance is an extremely powerful search tool for a variety of data sources, including names, addresses, medical records searching, genomic and disease research data, common misspellings, and more. Unlike fuzzy Hamming search, fuzzy edit distance is a more natural fuzzy search paradigm for many algorithms, since it does not require string matches to be of the same size.

The tradeoff is that fuzzy edit distance is not as amenable to full hardware parallelization, so algorithms reliant on it typically run slower than those that implement fuzzy Hamming search. In addition, the match string length added to the distance value must not exceed 32 bytes.

Date Search

The Date Search operation allows for exact date searches and for searching for dates within a range, for both structured and unstructured data, using the following date formats:

- `YYYY/MM/DD`
- `YY/MM/DD`
- `DD/MM/YYYY`
- `DD/MM/YY`
- `MM/DD/YYYY`
- `MM/DD/YY`

NOTE: The "/" character in the above list of formats can be replaced by any other single character delimiter. For example, YYYY-MM-DD and MM_DD_YYYY are both acceptable date formats.

Date searches extend the general relational expression defined previously, as follows:

```
(input_specifier relational_operator DATE(expression))
```

Different date ranges can be searched for by modifying the expression provided. There are two general expression types supported:

- `DATE(DateFormat operator ValueB)`
- `DATE(ValueA operator DateFormat operator ValueB)`

This is a full list of the supported expressions. `DateFormat` represents the format of the dates to search for and `ValueA` and `ValueB` represent dates to compare input data against.

- `DateFormat = ValueB`
- `DateFormat != ValueB (Not equals operator)`
- `DateFormat >= ValueB`
- `DateFormat > ValueB`
- `DateFormat <= ValueB`
- `DateFormat < ValueB`
- `ValueA <= DateFormat <= ValueB`
- `ValueA < DateFormat < ValueB`
- `ValueA < DateFormat <= ValueB`
- `ValueA <= DateFormat < ValueB`

For example, to find all dates after "02/28/12" in unstructured raw text, use the following search query criteria:

```
(RAW_TEXT CONTAINS DATE(MM/DD/YY > 02/28/12))
```

To find all matching dates between "02/28/12" and "01/19/15" but not including those two dates, in a record/field construct, where the field tag is date, use this:

```
(RECORD.date CONTAINS DATE(02/28/12 < MM/DD/YY < 01/19/15))
```

Time Search

Similar to the Date Search, the Time Search operation can be used to search for exact times or for times within a particular range, for both structured and unstructured data, using the following time formats:

- `HH:MM:SS`
- `HH:MM:SS:ss`

NOTE: The `'ss'` in the second format indicates hundredths of a second, and if specified should always be two digits.

Time searches extend the general relational expression defined previously as follows:

```
(input_specifier relational_operator TIME(expression))
```

Similar to the Date Search, different time ranges can be searched for by modifying the expression in the relational expression above. Again, there are two general expression types supported:

- `TIME(TimeFormat operator ValueB)`
- `TIME(ValueA operator TimeFormat operator ValueB)`

This is a list of the supported expressions. `TimeFormat` represents the format of the times to search for and `ValueA` and `ValueB` represent times to compare input data against.

- `TimeFormat = ValueB`
- `TimeFormat != ValueB` (Not equals operator)
- `TimeFormat >= ValueB`
- `TimeFormat > ValueB`
- `TimeFormat <= ValueB`
- `TimeFormat < ValueB`
- `ValueA <= TimeFormat <= ValueB`
- `ValueA < TimeFormat < ValueB`
- `ValueA < TimeFormat <= ValueB`
- `ValueA <= TimeFormat < ValueB`

For example, to find all times after "09:15:00", use the following search query criteria:

```
(RAW_TEXT CONTAINS TIME(HH:MM:SS > 09:15:00))
```

To find all matching times between "11:15:00" and "13:15:00" but not including those times in a record/field construct where the field tag is time, use:

```
(RECORD.time CONTAINS TIME(11:15:00 < HH:MM:SS < 13:15:00))
```

Number Search

The Number Search operation can be used to search for exact numbers or numbers in a particular range for both structured and unstructured input data.

The protocol rules are:

- No line breaks anywhere.
- The maximum number of characters is 64.
- Whitespace is not permitted within the number.
- The maximum number of consecutive digits is 16 base-10 characters. This means that numeric accuracy is compromised on numbers outside the range of approximately -253 and +253.
- The exponent value ranges when scientific notation is used are -512 and +512.
- Numeric separators are not permitted within a specified exponent.
- Infinity and "not a number" (NaN) are not accepted as numbers.

When a series of characters violates these parsing rules, the result will be the last valid sequence within the series, which allows for partial matches, and can be an excellent tool for analyzing potentially dirty data.

This general solution allows for numbers to be represented in arbitrary forms, including:

- Configurable separators, such as perhaps specifying a comma representing a thousands separator for the US number system (e.g., "7,000"), or specifying a dash separating fields in a phone number or a social security number (e.g., "1-800-555-1212" or "123-45-6789").
- Configurable decimals, such as perhaps specifying a period to represent the decimal for the US number system (e.g., "7.2").
- A minus sign to specify a negative value, such as "-7.2".
- Scientific notation, such as "-2e3", "-2e-3", "-2.2E+2", etc.
- Data results returned for a particular number matching specified criteria will be truncated after a total of 64 characters.

Number Searches extend the general relational expression defined previously as follows:

```
(input_specifier relational_operator NUMBER(expression, subitizer, decimal))
```

Different number ranges can be searched for by modifying the expression provided. There are two general expression types supported:

- `NUM operator1 "ValueA"`
- `"ValueA" operator1 NUM operator2 "ValueB"`

This is a full list of the supported expressions. `ValueA` and `ValueB` represent the numbers to compare the input data against.

- `NUM = "ValueA"`
- `NUM != "ValueA"` (Not equals operator)
- `NUM >= "ValueA"`
- `NUM > "ValueA"`
- `NUM <= "ValueA"`
- `NUM < "ValueA"`
- `"ValueA" <= NUM <= "ValueB"`
- `"ValueA" < NUM < "ValueB"`
- `"ValueA" < NUM <= "ValueB"`
- `"ValueA" <= NUM < "ValueB"`

The subitizer is defined as the separating character to use. For example, for standard US numbers, a comma would be specified. If other types of numbers are being searched, such as perhaps phone numbers, then a dash would be specified.

The decimal is defined as the decimal specifier to use. For example, for standard US numbers, a period would be specified.

Note that the subitizer and decimal must be different. If the same character is specified for both, an error message will be generated.

As an example of a fully qualified number search relational expression, to find all matching numbers using the US number system between but not including "1025" and "1050" in a record/field construct where the field tag is id, use:

```
(RECORD.id CONTAINS NUMBER("1025" < NUM < "1050", ",", "."))
```

The results will contain all numbers that are encountered in the input data that match the specified range. For example, if a scientific notation number "1.026e3" appears in the input data (which expands to "1,026"), then it will be reported as a match since it falls within the specified range. Similarly, the numbers 1049 and 1,049.9 would also match. However, the number -1,234 would not be a match, as it does not fall within the requested range, nor would the number +1,050.00001.

Currency Search

The Currency Search operation follows largely the same rules as the Number Search operation. The parsing protocol diagram changes slightly to allow for configurable currency identifiers, such as "$" for US currency. In addition, for currency, negative amounts are parsed using either the minus sign or parenthetical "()" notation.

Currency Searches are a type of number searches and extend the general relational expression defined previously as follows:

```
(input_specifier relational_operator CURRENCY(expression, currency, subitizer,
decimal))
```

Different currency ranges can be searched for by modifying the expression provided. There are two general expression types supported:

- `CUR operator1 "ValueA"`
- `"ValueA" operator1 CUR operator2 "ValueB"`

This is a list of the supported expressions. `ValueA` and `ValueB` represent the currency values to compare the input data against.

- `CUR = "ValueA"`
- `CUR != "ValueA"` (Not equals operator)
- `CUR >= "ValueA"`
- `CUR > "ValueA"`
- `CUR <= "ValueA"`
- `CUR < "ValueA"`
- `"ValueA" <= CUR <= "ValueB"`
- `"ValueA" < CUR < "ValueB"`
- `"ValueA" < CUR <= "ValueB"`
- `"ValueA" <= CUR < "ValueB"`

For currency searches, the subitizer is defined as the optional separating character that may be encountered when parsing currency. For example, for standard US currency, a comma would be specified. If other types of currencies are being searched, such as perhaps certain types of European currencies that use a period as the separator, a period would be specified.

The decimal is defined as the decimal specifier to use. For example, for standard US numbers, a period would be specified. For other currency types, an appropriate character would be specified.

Note that the subitizer and decimal specifier must be different. If the same character is specified for both, an error message will be generated.

As an example of a fully qualified currency search relational expression, to find all values using the US currency system between but not including "$450" and "$10,100.50" in a record/field construct where the field tag is price, use:

```
(RECORD.price CONTAINS CURRENCY("$450" < CUR < "$10,100.50", "$", ",", "."))
```

The results will contain all prices encountered in the input data that match the specified range. For example, if a price "$692.01" appears in the input data, then it will be reported as a match since it falls within the specified range. But currency values like -$123.00, $449.99 and $100,000 would not match.

IPv4 Search

The IPv4 Search operation can be used to search for exact IPv4 addresses or IPv4 addresses in a particular range, in both structured and unstructured text, using the standard "a.b.c.d" format for IPv4 addresses.

IPv4 Search extends the general relational expression defined previously as follows:

```
(input_specifier relational_operator IPV4(expression))
```

Different ranges can be searched for by modifying the expression in the relational expression above. There are two general expression types supported:

- `IP operator "ValueB"`
- `"ValueA" operator IP operator "ValueB"`

This is a list of supported expressions. `ValueA` and `ValueB` represent the IP addresses to compare the input data against.

- `IP = "ValueB"`
- `IP != "ValueB"` (Not equals operator)
- `IP >= "ValueB"`
- `IP > "ValueB"`
- `IP <= "ValueB"`
- `IP < "ValueB"`
- `"ValueA" <= IP <= "ValueB"`
- `"ValueA" < IP < "ValueB"`
- `"ValueA" < IP <= "ValueB"`
- `"ValueA" <= IP < "ValueB"`

For example, to find all IP addresses greater than 10.11.12.13, use the following search query criteria:

```
(RAW_TEXT CONTAINS IPV4(IP > "10.11.12.13"))
```

To find all matching IPv4 addresses adhering to 10.10.0.0/16 (that is, all IP addresses from 10.10.0.0 through 10.10.255.255 inclusive) in a record/field construct where the field tag is ipaddr, use:

```
(RECORD.ipaddr CONTAINS IPV4("10.10.0.0" <= IP <= "10.10.255.255"))
```

IPv6 Search

The IPv6 Search operation can be used to search for exact IPv6 addresses or IPv6 addresses in a particular range in both structured and unstructured text using the standard “a:b:c:d:e:f:g:h” format for IPv6 addresses. The double colon (::) is also supported, per RFC guidelines.

IPv6 searches extend the general relational expression defined previously, as follows:

```
(input_specifier relational_operator IPV6(expression))
```

Different ranges can be searched for by modifying the expression in the relational expression above. There are two general expression types supported:

- `IP operator "ValueB"`
- `"ValueA" operator IP operator "ValueB"`

This is list of supported expressions. `ValueA` and `ValueB` represent the IP addresses to compare the input data against.

- `IP = "ValueB"`
- `IP != "ValueB"` (Not equals operator)
- `IP >= "ValueB"`
- `IP > "ValueB"`
- `IP <= "ValueB"`

- `IP < "ValueB"`
- `"ValueA" <= IP <= "ValueB"`
- `"ValueA" < IP < "ValueB"`
- `"ValueA" < IP <= "ValueB"`
- `"ValueA" <= IP < "ValueB"`

For example, to find all IP addresses greater than `1abc:2::8`, use the following search query criteria:

```
(RAW_TEXT CONTAINS IPV6(IP > "1abc:2::8"))
```

To find all matching IPv6 addresses between “10::1” and “10::1:1”, inclusive, in a record/field construct where the field tag is “ipaddr6” use this:

```
(RECORD.ipaddr6 CONTAINS IPV6("10::1" <= IP <= "10::1:1"))
```

4

4: Command Line Tool

The `ryftrest` command line tool is a simple bash script with syntax that is very similar to the native `ryftprim` tool, and uses the `ryft-server` as a backend. The install package places the `ryftrest` tool in the "/usr/bin" directory.

There are some advantages to using `ryftrest` instead of `ryftprim` on the command line. The `ryftrest` tool:

- Can run complex queries. For example, combine fuzzy edit search with a date range and a time range.
- Can run in cluster mode or in single server mode.
- Runs on remote/distributed servers.
- Changes output to JSON, on the fly; a very useful functionality when used in conjunction with `jq`, the JSON command line processor.

This command will print extracted list of date strings.

```
ryftrest -q '(RECORD.id CONTAINS "100310")' -f '*.pcrime' --local --format=xml --
fields=ID,Date | jq ".results[].Date"
```

Parameters

Use the `ryftrest --help` for detailed syntax:

Parameter	Type	Description
Search Specific Parameters		
`-help | --help`		Print the help message
`-p | --mode`	String	Specify the search mode to run. Options are: • exact_search or es • fuzzy_hamming_search or fhs. Default • fuzzy_edit_distance_search or feds • date_search or ds • time_search or ts

Parameter	Type	Description
		• numeric_search or ns An empty or missing mode will default to `fhs` for common cases. However, if the query contains `DATE` or `TIME` keywords, then `ds` or `ts` will be used, respectively.
`-f | --file`	String	Specify an input filename.
`-i`		Specify case-insensitive analysis for supported primitives.
`-n | --nodes`	Integer	Specify 1-4 RCAB processing nodes to use.
`-d | --fuzziness`	Uint8	Specify the fuzzy search distance.
`-w | --width`	Integer	Specify the surrounding width.
`-s` `-q | --query`	String	Specify the search/query expression to use with `*_search` primitives.
`-od | --data`	String	Specify a data results file.
`-oi | --index`	String	Specify an index results file.
REST Specific Parameters		
`-a | --address`	String	Specify the ryft-server address. By default, http://localhost:8765
`--search`		Use /search endpoint (used by default) to print all found items.
`--count`		Use /count endpoint instead of /search to print just statistics.
`--limit=N`	Integer	Specifies the limit on total number of records printed (used with `/search`).
`--local`		Specify a local search. Opposite to --cluster.
`--cluster`		Specify a cluster search. Opposite to --local. Used by default.
`--format`	String	Specify format of the result records. Options are:

Parameter	Type	Description
		• raw - base-64 encoded data, by default. • xml - decode XML records. • json - decode JSON objects. • utf8 - for text search results are utf-8 string instead of base-64 encoded raw bytes.
`--fields`		Specifies comma-separated list of fields to return. Useful with XML and JSON formats.
`--no-stats`		Disable statistics output.
`--stream`		Use stream output format. Provides a sequence of JSON "tag-object" pairs to be able to decode input data on the fly (used for node communication within cluster).
`-v | --verbose`		Tell curl to be verbose.
`-vv | --pretty`		Specify properly indented formatting with jq (json).

Here are some examples of the command and parameters:

- Search and print all occurrences of "Joe" in text files:

```
/usr/bin/ryftrest -q=Joe -f=*.txt -vv
```

  Notice that this example shows "`-q=Joe`" instead of "`-q 'Joe'`" as the search value (as shown in the next 2 examples). The `ryftrest` command supports both methods.

- Print the number of matches and some performance numbers:

```
/usr/bin/ryftrest -q 'Joe' -f '*.txt' -vv --count
```

- Launch a structured search in "*.pcrime" files:

```
/usr/bin/ryftrest -q '(RECORD.id CONTAINS "100310")' -f '*.pcrime' --format=xml
--fields=ID,Date -vv
```

Search Mode Parameter

The `ryft-server` supports the following parameters for a new `mode`:

- `es` – exact search
- `fhs` – fuzzy hamming search
- `feds` – fuzzy edit distance search
- `ds` – date search
- `ts` – time search
- `ns` – numeric search

- `ipv4` – IPv4 search
- `ipv6` – IPv6 search

If the search mode is empty or missing, it defaults to `fhs`, fuzzy hamming search, with one exception. If the query contains the keywords “DATE” or “TIME” then `ds` or `ts` mode will be used, respectively.

Examples

Here are examples that show the similarities of using `ryftprim` vs. `ryftrest` on the command line.

Fuzzy Hamming Search

```
ryftprim -p fhs -q '(RECORD.desc CONTAINS "Jones")' -f mychicagoJones.pcrime -d 1 –v
```

The same command using `ryftrest`, with the additional options of running on local sever and selection of fields to return.

```
ryftrest -p fhs -q '(RECORD.desc CONTAINS "Jones")' -f mychicagoJones.pcrime -d 1 -v
--local --format=xml --fields=ID,Description
```

Fuzzy Edit Distance

```
ryftprim -p feds -q '(RECORD.desc CONTAINS "Jones")' -f mychicagoJones.pcrime -d 1 –v
```

The same command using `ryftrest` with the additional options of running on the cluster and selection of fields to return.

```
ryftrest -p feds -q '(RECORD.desc CONTAINS "Jones")' -f mychicagoJones.pcrime -d 1 -v
--cluster --format=xml --fields=ID,Description
```

Date Search

```
ryftprim -p ds -q '(RECORD.date CONTAINS DATE(MM/DD/YYYY > 04/13/2015))' -f
mychicago.pcrime –v
```

The same command using `ryftrest` with the additional options of running on local sever and executing the command to a different server.

```
ryftrest -p ds -q '(RECORD.date CONTAINS DATE(MM/DD/YYYY > 04/13/2015))' -f
mychicago.pcrime -v --local --format=xml --address "http://ryftone-0008:8765 "
```

Time Search

```
ryftprim -p ts -q '(RECORD.date CONTAINS TIME(HH:MM:SS > 10:20:00))' -f
mychicago.pcrime –v
```

The same command using `ryftrest` with the additional options of running on local sever and format the results as XML.

```
ryftrest -p ts -q '(RECORD.date CONTAINS TIME(HH:MM:SS > 10:20:00))' -f
mychicago.pcrime -v --local --format=xml
```

Complex Query Decomposition

One advantage of using `ryftrest` is the added benefit of query decomposition, which is not supported by `ryftprim`.

Query decomposition combines a few subqueries of the same type to minimize Ryft hardware calls. Here's the basic query structure of three query statements combined using two AND operators:

```
(RECORD CONTAINS DATE(...)) AND (RECORD CONTAINS DATE(...)) AND (RECORD CONTAINS
"sometext")
```

This is translated to just two calls, because the first two subqueries have the same date search type.

- `(RECORD CONTAINS DATE(...)) AND (RECORD CONTAINS DATE(...))`
- `(RECORD CONTAINS "sometext")`

Here is an example of the original search that looks for any record where the ID contains "1003".

```
ryftrest -q '(RECORD.id CONTAINS "1003")' -f mychicago.pcrime -v --local --format=xml
--fields=ID,Date,Description
```

And here is how it can be decomposed using the output of the first query as the input of the second query:

- Take the same search and decompose it to narrow down the results so that it only returns data where the date also contains 04/14/2015 :

  ```
  ryftrest -q '(RECORD.id CONTAINS "1003") AND (RECORD.date CONTAINS
  DATE(MM/DD/YYYY > 04/14/2015))' -f mychicago.pcrime -v --local --format=xml --
  fields=ID,Date,Description
  ```

- And then narrow it down even more by taking those results and using it as input to only return data where date also contains a time greater than 08:30:00:

  ```
  ryftrest -q '(RECORD.id CONTAINS "1003") AND (RECORD.date CONTAINS
  DATE(MM/DD/YYYY > 04/14/2015)) AND (RECORD.date CONTAINS TIME(HH:MM:SS >
  08:30:00))' -f mychicago.pcrime -v --local --format=xml --
  fields=ID,Date,Description
  ```

The OR Operator

The `OR` operator is supported, with the following caveats:

- Duplicate records are possible - duplicates are not discarded.
- Large data files may be copied - two or more intermediate result files are copied into one.

Using the `OR` operator with complex queries or with multiple `OR` operators requires additional steps.

In this scenarios, we have "A OR B" query and we have to call search two times: for "A" and "B" expressions with the same input data. Once the search is done, we would have two result files: "A-result.dat" and "B-result.dat" that needs to be combined into one. You need to then create the "result.dat" file and copy the contents of "A-result.dat" and "B-result.dat" into the one file. That operation could impact overall system performance, especially for complex queries containing multiple OR operators.

Example Input File

Here is an example input file “or.pcrime” that looks like this:


```
<rec><ID>10034183</ID><CaseNumber>HY223673</CaseNumber><Date>04/10/2015 10:15:00
PM</Date><Block>002XX</Block><IUCR>0486</IUCR><PrimaryType>BATTERY</PrimaryType><Descriptio
n>John</Description><LocationDescription>STREET</LocationDescription><Arrest>false</Arrest>
<Domestic>true</Domestic><Beat>0313</Beat><District>003</District><Ward>20</Ward><Community
Area>42</CommunityArea><FBICode>08B</FBICode><XCoordinate>1181263</XCoordinate><YCoordinate>
1863965</YCoordinate><Year>2015</Year><UpdatedOn>04/22/2015 12:47:10
PM</UpdatedOn><Latitude>41.781961688</Latitude><Longitude>-
87.610984705</Longitude><Location>"(41.781961688, -87.610984705)"</Location></rec>
<rec><ID>10034188</ID><CaseNumber>HY223687</CaseNumber><Date>04/10/2015 10:30:00
PM</Date><Block>003XX</Block><IUCR>0820</IUCR><PrimaryType>THEFT</PrimaryType><Description>
Jonny</Description><LocationDescription>SIDEWALK</LocationDescription><Arrest>false</Arrest
><Domestic>true</Domestic><Beat>1123</Beat><District>011</District><Ward>28</Ward><Communit
yArea>27</CommunityArea><FBICode>06</FBICode><XCoordinate>1152292</XCoordinate><YCoordinate>
1901795</YCoordinate><Year>2015</Year><UpdatedOn>04/22/2015 12:47:10
PM</UpdatedOn><Latitude>41.886390821</Latitude><Longitude>-
87.716204071</Longitude><Location>"(41.886390821, -87.716204071)"</Location></rec>
<rec><ID>10034213</ID><CaseNumber>HY223716</CaseNumber><Date>04/11/2015 10:45:00
PM</Date><Block>001XX</Block><IUCR>0470</IUCR><PrimaryType>PUBLIC PEACE
VIOLATION</PrimaryType><Description>Jenny</Description><LocationDescription>ALLEY</Location
Description><Arrest>true</Arrest><Domestic>false</Domestic><Beat>0522</Beat><District>005</
District><Ward>9</Ward><CommunityArea>53</CommunityArea><FBICode>24</FBICode><XCoordinate>11
77304</XCoordinate><YCoordinate>1825999</YCoordinate><Year>2015</Year><UpdatedOn>04/22/201
5 12:47:10 PM</UpdatedOn><Latitude>41.67786846</Latitude><Longitude>-
87.626642113</Longitude><Location>"(41.67786846, -87.626642113)"</Location></rec>
<rec><ID>10034327</ID><CaseNumber>HY223684</CaseNumber><Date>04/11/2015 10:53:00
PM</Date><Block>075XX</Block><IUCR>0486</IUCR><PrimaryType>BATTERY</PrimaryType><Descriptio
n>Lenny</Description><LocationDescription>RESIDENTIAL YARD
(FRONT/BACK)</LocationDescription><Arrest>false</Arrest><Domestic>true</Domestic><Beat>062
3</Beat><District>006</District><Ward>6</Ward><CommunityArea>69</CommunityArea><FBICode>08B
</FBICode><XCoordinate>1178866</XCoordinate><YCoordinate>1854896</YCoordinate><Year>2015</Y
ear><UpdatedOn>04/22/2015 12:47:10
PM</UpdatedOn><Latitude>41.757130314</Latitude><Longitude>-
87.620048394</Longitude><Location>"(41.757130314, -87.620048394)"</Location></rec>
<rec><ID>10034247</ID><CaseNumber>HY223708</CaseNumber><Date>04/12/2015 10:52:00
PM</Date><Block>081XX</Block><IUCR>0486</IUCR><PrimaryType>BATTERY</PrimaryType><Descriptio
n>Manny</Description><LocationDescription>VEHICLE NON-
COMMERCIAL</LocationDescription><Arrest>false</Arrest><Domestic>true</Domestic><Beat>0414<
/Beat><District>004</District><Ward>8</Ward><CommunityArea>46</CommunityArea><FBICode>08B</
FBICode><XCoordinate>1190898</XCoordinate><YCoordinate>1851594</YCoordinate><Year>2015</Yea
r><UpdatedOn>04/22/2015 12:47:10
PM</UpdatedOn><Latitude>41.747787125</Latitude><Longitude>-
87.57606016</Longitude><Location>"(41.747787125, -87.57606016)"</Location></rec>
<rec><ID>10034197</ID><CaseNumber>HY223707</CaseNumber><Date>04/12/2015 11:18:00
PM</Date><Block>001XX</Block><IUCR>1811</IUCR><PrimaryType>NARCOTICS</PrimaryType><Descript
ion>More</Description><LocationDescription>STREET</LocationDescription><Arrest>true</Arrest>
<Domestic>false</Domestic><Beat>1523</Beat><District>015</District><Ward>28</Ward><Communit
yArea>25</CommunityArea><FBICode>18</FBICode><XCoordinate>1141655</XCoordinate><YCoordinate>
1900379</YCoordinate><Year>2015</Year><UpdatedOn>04/22/2015 12:47:10
PM</UpdatedOn><Latitude>41.882708414</Latitude><Longitude>-
87.75530118</Longitude><Location>"(41.882708414, -87.75530118)"</Location></rec>
<rec><ID>10034248</ID><CaseNumber>HY223738</CaseNumber><Date>04/13/2015 11:48:00
PM</Date><Block>015XX</Block><IUCR>1121</IUCR><PrimaryType>DECEPTIVE
PRACTICE</PrimaryType><Description>Less</Description><LocationDescription>RESTAURANT</Locat
ionDescription><Arrest>false</Arrest><Domestic>false</Domestic><Beat>1012</Beat><District>0
10</District><Ward>24</Ward><CommunityArea>29</CommunityArea><FBICode>10</FBICode><XCoordina
```

```
te>1149938</XCoordinate><YCoordinate>1891833</YCoordinate><Year>2015</Year><UpdatedOn>04/22
/2015 12:47:10 PM</UpdatedOn><Latitude>41.859100084</Latitude><Longitude>-
87.725107817</Longitude><Location>"(41.859100084, -87.725107817)"</Location></rec>
<rec><ID>10037110</ID><CaseNumber>HY224060</CaseNumber><Date>04/13/2015 11:35:00
PM</Date><Block>011XX</Block><IUCR>0910</IUCR><PrimaryType>MOTOR VEHICLE
THEFT</PrimaryType><Description>No</Description><LocationDescription>STREET</LocationDescrip
tion><Arrest>false</Arrest><Domestic>false</Domestic><Beat>1211</Beat><District>012</Distri
ct><Ward>26</Ward><CommunityArea>24</CommunityArea><FBICode>07</FBICode><XCoordinate>115612
8</XCoordinate><YCoordinate>1907708</YCoordinate><Year>2015</Year><UpdatedOn>04/22/2015
12:47:10 PM</UpdatedOn><Latitude>41.902540073</Latitude><Longitude>-
87.701957503</Longitude><Location>"(41.902540073, -87.701957503)"</Location></rec>
<rec><ID>10034200</ID><CaseNumber>HY223668</CaseNumber><Date>04/14/2015 11:40:00
PM</Date><Block>054XX</Block><IUCR>0486</IUCR><PrimaryType>BATTERY</PrimaryType><Descriptio
n>John</Description><LocationDescription>RESIDENCE</LocationDescription><Arrest>false</Arre
st><Domestic>true</Domestic><Beat>1522</Beat><District>015</District><Ward>29</Ward><Commun
ityArea>25</CommunityArea><FBICode>08B</FBICode><XCoordinate>1140152</XCoordinate><YCoordina
te>1897108</YCoordinate><Year>2015</Year><UpdatedOn>04/22/2015 12:47:10
PM</UpdatedOn><Latitude>41.873760014</Latitude><Longitude>-
87.760900431</Longitude><Location>"(41.873760014, -87.760900431)"</Location></rec>
<rec><ID>10034234</ID><CaseNumber>HY223685</CaseNumber><Date>04/15/2015 11:30:00
PM</Date><Block>011XX</Block><IUCR>0320</IUCR><PrimaryType>ROBBERY</PrimaryType><Descriptio
n>Job</Description><LocationDescription>SIDEWALK</LocationDescription><Arrest>false</Arrest
><Domestic>false</Domestic><Beat>1824</Beat><District>018</District><Ward>42</Ward><Communi
tyArea>8</CommunityArea><FBICode>03</FBICode><XCoordinate>1175283</XCoordinate><YCoordinate>
1908223</YCoordinate><Year>2015</Year><UpdatedOn>04/22/2015 12:47:10
PM</UpdatedOn><Latitude>41.903544846</Latitude><Longitude>-
87.631582982</Longitude><Location>"(41.903544846, -87.631582982)"</Location></rec>
```


This file is based on “chicago.pcrime” with just 10 records and modified content.

Example Queries

Using the “or.pcrime” data file, above, consider these three sample queries:

1. To get all records, run:
   `(RECORD.id CONTAINS "1003")`
2. To get first 6 records, run:
   `(RECORD.date CONTAINS DATE(MM/DD/YYYY <= 04/12/2015))`
3. To get last 5 records, run:
   `(RECORD.date CONTAINS TIME(HH:MM:SS > 11:15:00))`

Combining queries number 2 and 3 with the `OR` operator returns 11 records (ID=10034197 will be included twice):

```
((RECORD.date CONTAINS DATE(MM/DD/YYYY <= 04/12/2015)) OR (RECORD.date CONTAINS
TIME(HH:MM:SS > 11:15:00)))
```

Combining queries number 1 and 3 returns 15 records (the last 5 will have duplicates):

```
(RECORD.id CONTAINS "1003") OR (RECORD.date CONTAINS TIME(HH:MM:SS > 11:15:00))
```

You can also could use both the `AND` and `OR` operators together, like this:

```
(RECORD.id CONTAINS "1003") AND ((RECORD.date CONTAINS DATE(MM/DD/YYYY <= 04/12/2015))
OR (RECORD.date CONTAINS TIME(HH:MM:SS > 11:15:00)))
```

The `OR` operator has lower priority, so using “OR b AND c” is equivalent to using “OR (b AND c)”:

```
(RECORD.date CONTAINS DATE(MM/DD/YYYY <= 04/12/2015)) OR (RECORD.date CONTAINS
TIME(HH:MM:SS > 11:15:00)) AND (RECORD.id CONTAINS "1003")
```

Minimize Output

To minimize output, use the `--count` option. Passing this flag makes the tool use the `/count` endpoint instead of `/search` so that only the search statistics are printed, not the search results.

```
ryftrest -q '(RECORD.id CONTAINS "100310")' -f '*.pcrime' --local --format=xml --
fields=ID,Date --count
{"matches":24,"totalBytes":1415539,"duration":1140,"dataRate":1.1841782352380585,"fabr
icDataRate":0}
```

An alternative option is to use `jq` command line JSON processor:

```
ryftrest -q '(RECORD.id CONTAINS "100310")' -f '*.pcrime' --local --format=xml --
fields=ID,Date | jq ".stats"
{
  "fabricDataRate": 6573.359375,
  "dataRate": 9.723904570178872,
  "duration": 676,
  "totalBytes": 6892667,
  "matches": 81
}
```

This command is almost equal to the previous one. Without the `--count` option, all data is still transferred. All processing is done on the client side.

Using `jq` it's possible to do advanced data processing. This command prints a list of matching date strings:

```
ryftrest -q '(RECORD.id CONTAINS "100310")' -f '*.pcrime' --local --format=xml --
fields=ID,Date | jq ".results[].Date"
"04/13/2015 11:45:00 PM"
"04/13/2015 11:40:00 PM"
"04/13/2015 11:30:00 PM"
"04/13/2015 11:25:00 PM"
"04/13/2015 11:20:00 PM"
"04/13/2015 11:18:00 PM"
"04/13/2015 11:10:00 PM"
"04/13/2015 11:00:00 PM"
"04/13/2015 11:35:00 AM"
"04/13/2015 11:00:00 AM"
"04/13/2015 11:53:00 PM"
"04/13/2015 11:45:00 PM"
"04/13/2015 11:40:00 PM"
"04/13/2015 11:30:00 PM"
"04/13/2015 11:25:00 PM"
"04/13/2015 11:20:00 PM"
"04/13/2015 11:18:00 PM"
"04/13/2015 11:10:00 PM"
"04/13/2015 11:00:00 PM"
```

Preserve Search Results

By default, all search results are deleted from the Ryft server once they are delivered to user. However, you can preserver your search results so that the output from the first search becomes the input for the second search.

Enable the "search in the previous results" feature using two query parameters:

- Output File: The `data=output.dat` or `-od output.dat` parameter keeps the search results on the Ryft server in the file /ryftone/output.dat. It is possible to use that file as an input for the subsequent search call `files=output.dat`.

  NOTE: It is important to use consistent file extension for the structured search in order to let Ryft use appropriate RDF scheme!
- Index File: The `index=index.txt` or `-oi index.txt` parameter keeps the search index under /ryftone/index.txt.

Here is an example:

```
ryftrest -q '(RECORD.id CONTAINS "100310")' -f '*.pcrime' --local --format=xml --
fields=ID,Date -od mytest.pcrime -oi mytest.txt
cat /ryftone/mytest.txt
```

The `data=` and `index=` query parameters are supported by both `/search` and `/count` endpoints.

In case of complex search expression, query decomposition saves data and index of the top subquery.

WARNING: If data or index file with the same name already exist, it will be overridden!

JSON Format Support

If the input file set contains JSON data, the found records could be decoded using the `format=json` query parameter.

```
ryftrest -q '(RECORD.Name CONTAINS "Bruce")' -f 'CitizensOfGotham.json' --local
--format=json
```

This is the output:

```
{"results":[{"Actors":[{"Name":"Adam West"},{"Name":"Michael Keaton"},{"Name":"Val
Kilmer"},{"Name":"George Clooney"},{"Name":"Christian Bale"}],"AlterEgo":"The
Batman","Name":"Bruce
Wayne","_index":{"file":"/CitizensOfGotham.json","offset":8,"length":1108,"fuzziness":
0,"host":"ryftone-310"}}
,{"Actors":[{"Name":"Adam West"},{"Name":"Michael Keaton"},{"Name":"Val
Kilmer"},{"Name":"George Clooney"},{"Name":"Christian Bale"}],"AlterEgo":"The
Batman","Name":"Bruce
Wayne","_index":{"file":"/CitizensOfGotham.json","offset":5688,"length":1108,"fuzzines
s":0,"host":"ryftone-310"}}
,{"Actors":[{"Name":"Adam West"},{"Name":"Michael Keaton"},{"Name":"Val
Kilmer"},{"Name":"George Clooney"},{"Name":"Christian Bale"}],"AlterEgo":"The
Batman","Name":"Bruce
Wayne","_index":{"file":"/CitizensOfGotham.json","offset":11368,"length":1108,"fuzzine
ss":0,"host":"ryftone-310"}}
],"stats":{"matches":3,"totalBytes":17040,"duration":652,"dataRate":0.0249242490054639
55,"fabricDataRate":0}
```

To minimize output or to get just a subset of fields, use the `fields=` query parameter. For example, to get AlterEgo and Name fields from the “CitizensOfGotham.json” file, pass this parameter: `format=json&fields=AlterEgo,Name`.

```
ryftrest -q '(RECORD.Name CONTAINS "Bruce")' -f 'CitizensOfGotham.json' --local --
format=json --fields=AlterEgo,Name
```

Here is the output:


```
{"results":[{"AlterEgo":"The Batman","Name":"Bruce
Wayne","_index":{"file":"/CitizensOfGotham.json","offset":8,"length":1108,"fuzziness":
0,"host":"ryftone-310"}}
,{"AlterEgo":"The Batman","Name":"Bruce
Wayne","_index":{"file":"/CitizensOfGotham.json","offset":5688,"length":1108,"fuzzines
s":0,"host":"ryftone-310"}}
,{"AlterEgo":"The Batman","Name":"Bruce
Wayne","_index":{"file":"/CitizensOfGotham.json","offset":11368,"length":1108,"fuzzine
ss":0,"host":"ryftone-310"}}
],"stats":{"matches":3,"totalBytes":17040,"duration":734,"dataRate":0.0221397961192949
6,"fabricDataRate":0}
```


NOTE: JSON format (just like XML) can only be used with structured search.