

# Revelytix Rex

## Quick Start Guide

This document describes the basic details on how to load and run Rex.

### Installation Overview

Rex is distributed as a zip file, named `rex-x.y.z-distribution.zip`, where `x.y.z` denotes the Rex version. From inside the directory where you want Rex installed, unzip the distribution file. This will create a directory named `rex-<version>`, e.g., `rex-0.2.0`. Under the main directory are the following directories:

- `bin` - command scripts to run Rex Tool
- `doc` - this document and others
- `examples` - demos and examples
- `lib` - product libraries
- `license` - product license information

Check the README file in the root of the installation for a detailed description of the contents of the distribution.

### Rule Programs

Rex loads and executes rules written in the Rules Interchange Format (RIF). For user accessibility, programs should be written in the RIF Presentation Syntax. For details, please see:

[http://www.w3.org/TR/2010/REC-rif-blid-20100622/#EBNF\\_Grammar\\_for\\_the\\_Presentation\\_Syntax\\_of\\_RIF-BLD\\_28Informative.29](http://www.w3.org/TR/2010/REC-rif-blid-20100622/#EBNF_Grammar_for_the_Presentation_Syntax_of_RIF-BLD_28Informative.29)

As well as the standard frame syntax for representing RDF triples (using the style of F-Logic), Rex also supports unary and binary predicates for triples. Unary predicates represent a type declaration, while binary predicates represent a triple, where the arguments represent the subject and object, respectively.

The following is an example document that determines *uncleOf* relationships based on the relationships of *childOf* and *brotherOf*.

```
Document (
  Base(<http://example.com/family#>)
  Prefix(ex <http://example.com/family#>)

  Group(
    Forall ?x ?y ?z (
      ex:uncleOf(?z ?x) :- And( ex:childOf(?x ?y) ex:brotherOf(?z ?y) )
    )
  )
)
```

File listing: *family.rp*

## Data Source and Storage

Rex is designed to process rules on RDF data obtained from one or more sources, and to store new entailments in a writable RDF store. The destination store may be the same store that is storing some or all of the original data.

Rex expects that all of the stores that it communicates with will support SPARQL 1.1 Query and Protocol. In addition, the destination store must also support SPARQL 1.1 Update.

An example system that supports all of these protocols is Fuseki, from Apache Jena. Fuseki can be found at:

<http://openjena.org/wiki/Fuseki#Download>

## Configuration

The easiest way to configure Rex is with a configuration file. The configuration is in RDF, and any valid RDF representation may be used. This document will represent RDF using Turtle.

A configuration file must contain the following:

- The location of a rule program.
- Connection details for the RDF database that will store the new entailments. This includes:
  - An endpoint URL that supports SPARQL 1.1 Update operations.
  - An endpoint URL for issuing SPARQL 1.1 Queries.
  - The URI of the graph that will store the entailments.
- The locations of the data to process with rules. This is a set of pairs of the following:
  - The URI of a graph containing data to be processed.
  - The endpoint URL where the graph can be queried.

Note that URLs are a type of URI that identifies the *location* of a resource. URIs are used as an

identifier for a resource, and may or may not describe a location.

# Vocabulary

## Prefixes

The RDF vocabulary for the Rex configuration uses the following namespaces:

rdf: [<http://www.w3.org/1999/02/22-rdf-syntax-ns#>](http://www.w3.org/1999/02/22-rdf-syntax-ns#)

rex: [<http://revelytix.com/rex#>](http://revelytix.com/rex#)

sd: [<http://www.w3.org/ns/sparql-service-description#>](http://www.w3.org/ns/sparql-service-description#)

## Classes

Classes used in the configuration are:

### rex:Config

A Rex configuration. Instances of this contain configuration data.

### rex:Destination

A description of where entailed data will be stored.

### sd:Service

A SPARQL service made available via the SPARQL Protocol.

See: <http://www.w3.org/TR/sparql11-service-description/#sd-Service>

### sd:Dataset

**subClassOf:** *sd:GraphCollection*

An RDF Dataset comprised of a default graph and zero or more named graphs.

See: <http://www.w3.org/TR/sparql11-service-description/#sd-Dataset>

### sd:GraphCollection

An RDF Dataset comprised of zero or more named graphs (no default graph).

See: <http://www.w3.org/TR/sparql11-service-description/#sd-GraphCollection>

### sd:NamedGraph

A graph with a name. Descriptions are optional and ignored.

See: <http://www.w3.org/TR/sparql11-service-description/#sd-NamedGraph>

## Properties

Properties in the configuration are:

**rex:rulePath**

**domain:** *rex:Config*

**range:** *xsd:string*

A path to a rule program document. This may be absolute, or relative to the running system. Note that a path may contain system specific elements, such as the path separator. This string can also be a URL. Do not use in conjunction with *rex:ruleURL*.

**rex:ruleURL**

**domain:** *rex:Config*

The URL of a rule document. May be any standard kind of URL. e.g. *file:* or *http:*. Do not use in conjunction with *rex:rulePath*.

**rex:destination**

**domain:** *rex:Config*

**range:** *rex:Destination*

The description of the location where entailments will be stored.

**rex:jenaEndpoint**

**domain:** *rex:Destination*

The base URL for a Jena/Fuseki endpoint. The paths */query* and */update* will be appended to access the query and update URLs. This may be used in place of the combination of *rex:queryUrl* and *rex:updateUrl* that is usually required.

**rex:queryUrl**

**domain:** *rex:Destination*

The URL of a service that can process SPARQL 1.1 queries. Used to retrieve entailed data.

**rex:updateUrl**

**domain:** *rex:Destination*

The URL of a service that can process SPARQL 1.1 updates. Used to store entailed data.

**rex:graph**

**domain:** *rex:Destination*

The URI identifying a graph.

**rex:endpointGraphs**

**domain:** *rex:Config*

**range:** *rdf:List (sd:Service)*

Description of the data to be processed by the rules program. All elements of the list

must be instances of *sd:Service*.

### **sd:endpoint**

**domain:** *sd:Service*

URL of a SPARQL 1.1 query service.

See: <http://www.w3.org/TR/sparql11-service-description/#sd-endpoint>

### **sd:dataset**

**domain:** *sd:Service*

**range:** *sd:Dataset*

An RDF dataset. This refers to a default graph and a set of zero or more named graphs.

Use this when you need to access the default graph on an endpoint.

### **sd:graphCollection**

**domain:** *sd:Service*

**range:** *sd:GraphCollection*

An RDF graph collection. This is the same as using *sd:dataset* with an empty default graph. Use this when you do not want to include the default graph on an endpoint.

### **sd:namedGraph**

**domain:** *sd:GraphCollection*

**range:** *sd:NamedGraph*

A graph with a URI as a name.

See: <http://www.w3.org/TR/sparql11-service-description/#sd-namedGraph>

### **sd:name**

**domain:** *sd:NamedGraph*

The URI of the graph name.

See: <http://www.w3.org/TR/sparql11-service-description/#sd-name>

## **Example**

### **Example 1**

The following is an example document for performing entailments on data found in 3 graphs.

The rule document is on the UNIX path of: */tmp/rex.rp*

The source graphs are at:

Endpoint: <http://alpha.example.com/dataset/query> graph: <http://example.com/data>

graph: <http://example.com/info>

graph: default

Endpoint: <http://beta.example.com/dataset/query> graph: <http://example.com/external>

The destination is a server on the local system. The URLs for service access are:

Query: <http://localhost:3030/dataset/query>

Update: <http://localhost:3030/dataset/update>

The output graph is: <http://example.com/entailed>

```
@prefix sd:    <http://www.w3.org/ns/sparql-service-description#> .
@prefix rex:  <http://revelytix.com/rex#> .
@prefix :    <http://example.com/>

:config rex:rulePath "/tmp/rex.rp" ;
  rex:destination [
    rex:queryUrl <http://localhost:3030/dataset/query> ;
    rex:updateUrl <http://localhost:3030/dataset/update> ;
    rex:graph :entailed
  ] ;
  rex:endpointGraphs (
    [ sd:endpoint <http://alpha.example.com/dataset/query> ;
      sd:dataset [
        sd:namedGraph [ sd:name :data ] ;
        sd:namedGraph [ sd:name :info ]
      ] ]
    [ sd:endpoint <http://beta.example.com/dataset/query> ;
      sd:graphCollection [ sd:namedGraph [ sd:name :external ] ] ]
  ) .
```

File listing: *config1.ttl*

## Example 2

A much simpler configuration is where the input data and the output are all in the same local store. In this example, the destination is known to be a Jena/Fuseki server, and so the *rex:jenaEndpoint* property can be used. The result is an identical endpoint to the one described in example 1. The rule document is also described with a URL to make it more portable. Finally, the graph URIs use an informal scheme, which is sometimes useful for local data.

The rule document is at the local URL of: *file:///tmp/rex.rp*

The source graphs are local in the graph: *input:data*

The destination is on the local system in the graph: *output:data*

```
@prefix sd:    <http://www.w3.org/ns/sparql-service-description#> .
```

```
@prefix rex: <http://revelytix.com/rex#> .
@prefix : <http://example.com/>

:config rex:ruleUrl <file:///tmp/rex.rp> ;
  rex:destination [
    rex:jenaEndpoint <http://localhost:3030/dataset> ;
    rex:graph <output:data>
  ] ;
  rex:endpointGraphs (
    [ sd:endpoint <http://localhost:3030/dataset/query> ;
      sd:graphCollection [ sd:namedGraph [ sd:name <input:data> ] ] ]
  ) .
```

File listing: *config1.ttl*

## Running Rex

Rex Beta 0.2 is configured for access through a Java API or as a command line tool.

### Command line

Ensure that the Rex jar is available in the class path. The command to run rules with a given configuration file called *configuration.ttl* is:

```
java revelytix.rex.Rules configuration.ttl
```

The configuration parameter may be a relative or absolute path to the configuration file.

### API

Rex provides the following static methods on the class **revelytix.rex.Rules** which is found in the Rex Jar file.

```
void run(java.io.File config)
  throws java.io.FileNotFoundException, Exception
```

Runs the rules found in the *config* file.

**params:**     *config*: An RDF Rex configuration file.  
**throws:**    *FileNotFoundException*: if the config file does not exist.  
              *Exception*: Due to an error processing the rules.

```
void run(String config)
    throws java.io.FileNotFoundException, Exception
```

Runs the rules found in *config*. The *config* argument may be a path to a file or a URL for a document.

**params:** *config*: Description of an RDF Rex configuration document. Detects paths or URLs automatically

**throws:** *FileNotFoundException*: if the config describes a file that does not exist.  
*Exception*: Due to an error processing the rules.

```
java.util.Map<Object, Object> readConfig(java.io.File config)
    throws FileNotFoundException, Exception
```

Reads a configuration file and returns a nested map structure representing the entire configuration. Useful for determining if parameters are being read correctly.

**params:** *config*: An RDF Rex configuration file.

**throws:** *FileNotFoundException*: if the config file does not exist.  
*Exception*: Due to an error reading the configuration.

```
java.util.Map<Object, Object> readConfig(String config)
    throws FileNotFoundException, Exception
```

Reads a configuration file and returns a nested map structure representing the entire configuration. Useful for determining if parameters are being read correctly. The *config* argument may be a path to a file or a URL for a document.

**params:** *config*: Description of an RDF Rex configuration document. Detects paths or URLs automatically

**throws:** *FileNotFoundException*: if the config describes a file that does not exist.  
*Exception*: Due to an error reading the configuration.

```
void load(String endpoint, String graphName, String file)
    throws Exception
```

Convenience method to load data into an RDF store.

**params:** *endpoint*: The URL of the endpoint of the store to load data into.  
*graphName*: The URI of the graph to load data into.  
*file*: The file containing an RDF document to be loaded into the store.

**throws:** *Exception*: Due to an error loading the data.

```
void drop(String endpoint, String graphName)
    throws Exception
```

Convenience method to drop a graph from an RDF store.

**params:**     *endpoint*: The URL of the endpoint of the store to drop data from.  
              *graphName*: The URI of the graph to drop data from.

**throws:**     *Exception*: Due to an error removing the data.

## Step-by-Step Example

The following provides a step-by-step guide to set up a simple rule program running on data from scratch.

Rex requires a Java Virtual Machine (JVM), version 1.5 or higher. This tutorial requires Java 6, to support the suggested RDF store.

### Installing an RDF Store

Rex requires an RDF store that supports SPARQL Update/Query 1.1. It also requires at least one store holding the data to be processed that supports SPARQL Query. These can easily be the same store.

One store that supports these features is Fuseki. You should download a recent zip file from:

<http://openjena.org/wiki/Fuseki#Download>

As of this writing, the most recent version was Fuseki-0.2.0. Unzip the archive and open a command prompt in the resulting directory.

On UN\*X style systems (Linux, Mac OS X, etc) there is a script to start a server. To start a server that supports updates on an endpoint called "dataset", execute the following:

```
./fuseki-server --update --desc tdb.ttl /dataset
```

On Windows, the same configuration can be executed with:

```
java -Xmx1200M -jar fuskei-sys.jar --update --desc tdb.ttl /dataset
```

## Loading Data

The data in this example is a simple Turtle file, called *family.ttl*, containing the following:

```
@prefix ex: <http://example.com/family#> .  
  
ex:Alice ex:childOf ex:Bruce .  
ex:Chuck ex:brotherOf ex:Bruce .
```

File listing: *family.ttl*

There are several ways to load this data, but the simplest is to simply use the web interface on Fuseki. After starting Fuseki in the previous section, there should now be a service running on port 3030. You can access this by directing a browser to the URL:

<http://localhost:3030/>

Select the top link, which is labelled “Control Panel”. This goes to the Control Panel, which lets you select the the dataset that was configured as “/dataset”. After selecting the dataset, the browser will display the Fuseki Query screen, as shown in Figure 1.

## Fuseki Query

Dataset: /dataset

---

### SPARQL Query

Output:

XSLT style sheet (blank for none):

Force the accept header to text/plain regardless

---

### SPARQL Update

---

### File upload

File:  No file chosen

Graph:

Figure 1

Press the “Choose File” button at the bottom of the page, and select the *family.ttl* file.

This screen places restrictions on the form of the URI of the graph, so in this example use the

name “<http://example.com/family>”. These selections should appear similar to figure 2.

### File upload

File:  family.ttl  
 Graph:

Figure 2

After pressing the “Upload” button the browser should report that two triples were loaded.

## The Rule Program

The program to run on the data is the *family.rp* program presented earlier in this document. This program entails an *uncleOf* relationship base on *childOf* and *brotherOf*.

Of course, many more entailments are also possible, such as *parentOf* and *siblingOf*, and if gender types were introduced then more information could be determined (son, daughter, nephew, neice, etc). However, for the sake of this example, the program will stick to the simple *uncleOf* relationship.

The *family.rp* program is reproduced here:

```
Document (
  Base(<http://example.com/family#>)
  Prefix(ex <http://example.com/family#>)

  Group(
    Forall ?x ?y ?z (
      ex:uncleOf(?z ?x) :- And( ex:childOf(?x ?y) ex:brotherOf(?z ?y) )
    )
  )
)
```

File listing: *family.rp*

## Configuring Rex

The data is now ready to be processed on the local host, through the Fuseki service <http://localhost:3030/dataset> and with from the graph named <http://example.com/family>.

The program file *family.rp* should be stored in the current directory for this example (it will be referred to with a relative path). Finally, the output will be sent to <http://example.com/entailed>. The configuration file for all of this is shown here:

```

@prefix sd:    <http://www.w3.org/ns/sparql-service-description#> .
@prefix rex:  <http://revelytix.com/rex#> .
@prefix :     <http://example.com/config> .

:config
  rex:rulePath "family.rp";

  rex:destination [
    rex:jenaEndpoint <http://localhost:3030/dataset> ;
    rex:graph <http://example.com/entailed>
  ] ;
  rex:endpointGraphs (
    [ sd:endpoint <http://localhost:3030/dataset/query> ;
      sd:graphCollection [
        sd:namedGraph [ sd:name <http://example.com/family> ]
      ]
    ]
  ) .

```

File listing: *family-cfg.ttl*

## Running

After configuring all the files as shown, make sure that the Rex Jar file, the configuration file *family-cfg.ttl* and the rule program *family.rp* are all in the current directory. The program can be executed with the command:

```
java -classpath rex-0.2.0.jar revelytix.rex.Rules family-cfg.ttl
```

### Results

To see the results of the program, you can select the data from the output graph using the Fuseki Query screen.

In the SPARQL Query field enter the following query:

```
SELECT ?s ?p ?o
WHERE { GRAPH <http://example.com/entailed> { ?s ?p ?o } }
```

You should get the result:

s	p	o
<http://example.com/family#Chuck>	<http://example.com/family#uncleOf>	<http://example.com/family#Alice>

To see the complete results, ask for the union of all graphs (this may be restricted to just the

required graphs if there are other graphs available):

```
SELECT ?s ?p ?o  
WHERE { GRAPH ?g { ?s ?p ?o } }
```

<b>s</b>	<b>p</b>	<b>o</b>
<http://example.com/family#Chuck>	<http://example.com/family#brotherOf>	<http://example.com/family#Bruce>
<http://example.com/family#Alice>	<http://example.com/family#childOf>	<http://example.com/family#Bruce>
<http://example.com/family#Chuck>	<http://example.com/family#uncleOf>	<http://example.com/family#Alice>