CONTENTS

1 Overview and features 1
2 Installation and Migration 3
3 Loading and Saving Data Using Refine 13
4 RDF-izing Tabular Data 17
5 Ontotext Refine CLI 29
6 Installing Refine Extensions 41
7 Release Notes 45
8 Additional Resources 49
Ontotext Refine ("Refine") is a version of the open-source OpenRefine data transformation tool adapted to work with Ontotext GraphDB.

Refine allows fast cleaning, mapping and transformation of any structured data to RDF and loading it to GraphDB.

It supports input from:
- Tabular formats (TSV, CSV, *SV)
- Fixed-width text files
- Excel (XLS, XLSX)
- JSON, JSON-LD, XML
- RDF: XML, Turtle/N3
- Databases (PostgreSQL, MySQL, MariaDB, SQLite)

You can input data from local files, remote URLs, and clipboard snippets.

Refine enables you to:
- Create projects and upload your data file(s)
- Clean and transform the data using powerful row and column manipulations, faceting, clustering
- Implement complex transformations using:
  - Expressions and GREL (Google Refine Expression Language)
  - GraphDB Functions including SPIN functions
  - Combining datasets between Refine projects by using the cross() function.
  - Combining multiple repositories and projects using SPARQL Federation and the virtual SPARQL endpoint of each Refine project
  - the Refine command line interface
- Create a visual RDF mapping of the cleaned data
  - The RDF mapping visual UI is optimized to guide you in defining URLs, choosing the right predicates and types, defining datatypes, etc.
  - Generate the respective SPARQL query
  - Export the RDF data
- Expose a virtual SPARQL endpoint that allows you to write complex SPARQL queries
  - Export RDF data using a SPARQL CONSTRUCT query
  - Load RDF data to a GraphDB repository using Federated SPARQL UPDATE query
- Export project configurations and mappings in order to automate a transformation on more data of a similar structure
You can generate Refine queries from semantic models using the open source rdf2rml toolkit.
2.1 Installation

2.1.1 Requirements

It is recommended to run Refine with:

- At least 1 gigabyte of memory.
- At least 256 megabytes of disk space.
- Java Runtime Environment or Java SE Development Kit versions between 11 and 15
  - The recommended JRE is Temurin-11.0.15+10.
  - Java is not required for native installations, which include that JRE
  - We recommend using Refine with 64-bit version of Java

**Note:** Refine is an in-memory application, meaning that it loads the full data of the current project in memory. Disk storage is needed for input files, autosave of the project and its metadata (every minute), and exported files.

If you have large data, you should provide more memory. We can define large data as:

- More than 800,000 total cells.
- An input size of more than 50 megabytes.
- More than 50 rows per record.

<table>
<thead>
<tr>
<th>Number of cells</th>
<th>Minimal memory</th>
<th>Optimal memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>400,000</td>
<td>512MB</td>
<td>1GB</td>
</tr>
<tr>
<td>800,000</td>
<td>1.5GB</td>
<td>3GB</td>
</tr>
<tr>
<td>2,000,000</td>
<td>3GB</td>
<td>6GB</td>
</tr>
<tr>
<td>4,000,000</td>
<td>6GB</td>
<td>12GB</td>
</tr>
<tr>
<td>6,000,000</td>
<td>8GB</td>
<td>16GB</td>
</tr>
<tr>
<td>8,000,000</td>
<td>10GB</td>
<td>20GB</td>
</tr>
</tbody>
</table>

As a rule of thumb, optimal memory grows by 2GB for each million cells.

You can run Refine with lower memory, but you are risking an Out of Memory (OOM) exception and we can’t recommend such a deployment.

**Warning:** Resource usage is cumulative for all projects that are created or opened in this session. For example, if you try to open two projects, one with 2,000,000 cells and another with 4,000,000 cells, you should have 8GB of memory.
If you use the `cross()` function in a project, the other project is also opened. This function is useful for combining datasets between Refine projects.

### 2.1.2 Running Refine

Refine can be operated as a desktop or a server application. Choose the one that best suits your needs and follow the steps below to run Refine:

- **As a desktop (native) installation:** For desktop users who want to test out the application. It is very straightforward, sets up an uninstaller, and desktop shortcuts.
- **As a standalone server:** Recommended for production. Comes packaged with a preconfigured web server. It offers greater configurability.
- **In a Docker container:** A stable way to have an installation that “just works”, without having to worry about the specifics of your operating system. See our images on Dockerhub.

### 2.1.3 Native Launchers

The easiest way to set up and run Refine is to use the native desktop installations. This does not require the use of a console and works with a graphic user interface. For this installation approach, you do not need to download Java, as the application comes with JDK 11 bundled.

Go to the Refine download page and request your copy. You will receive an email with the download link. According to your OS, proceed as follows:

**Windows**

1. Download the Refine .msi installer file.
2. Double-click the application file and follow the on-screen installer prompts.
3. You will be asked for an installation location.
4. Locate the application in the Windows Start menu or on the desktop and start it.

**MacOS**

1. Download the Refine .dmg file.
2. Double-click it and get a virtual disk on your desktop.
3. Copy the program from the virtual disk to your hard disk Applications folder.
4. Locate the application on the desktop and start it.
1. Download the Refine .deb or .rpm file.
2. Install the package:
   • Debian and derivatives: sudo dpkg -i <package-name>.deb
   • Redhat and derivatives: sudo rpm -i <package-name>.rpm
   • CentOS and derivatives: sudo yum install <package-name>
3. Alternatively, double-click the package.
4. Locate the application on the desktop and start it.

Configuring Native Refine

Native Refine uses two configuration files:

• {ontorefine.dist}/Ontotext Refine.cfg are Java launcher properties. See JVM Configuration for details.
  – On Windows, this is \Users\<username>\AppData\Local\Ontotext Refine\app\Ontotext Refine.cfg
• {user.home}/OntoRefine/ontorefine.properties are Refine properties. See Refine Configuration Parameters for details.
• On Windows, this is \Users\<username>\AppData\Roaming\OntoRefine\ontorefine.properties
• You can change this location using the Java property ontorefine.config.path or environment variable ONTOREFINE_CONFIG_PATH.

Once you have started the native application, a small icon appears in the statusbar/menu/tray area. It allows you to check if the application is running, stop it or change the configuration settings. Additionally, an application window is opened, where you can open the Refine web app, go to the Refine documentation, configure settings, and see all log files. You can reopen this window by choosing Show Refine window from the statusbar icon menu.
In addition, a few of the parameters (for connecting to GraphDB) can be set in the Refine web app.

**Stopping Native Refine**

To stop the application, simply quit it from the statusbar/menu/tray area icon, or close the Refine application window.

**Hint:** On some Linux systems, there is no support for statusbar/menu/tray area. If you have hidden the Refine window, you can quit it by killing the process.

### 2.1.4 Standalone Server

The recommended way of running Refine in production use cases is as a standalone server. The server is platform-independent and includes all recommended JVM parameters for immediate use.

**Note:** The standalone distribution does not package a JDK like the native launcher. Before running Refine, please make sure to have JDK or JRE installed.
Running Standalone Refine

1. Download the Refine distribution archive and unpack it.
2. Start Refine by executing the\texttt{ontorefine} startup script located in the \texttt{bin} directory of the Refine distribution.

A message appears in the console telling you that Refine has been started. You can access the Refine client at \url{http://localhost:7333/} in your browser.

Configuring Refine Server

See \textit{Refine Configuration Parameters} for a description of Refine settings.

2.1.5 Docker

In addition to \textit{Standalone Server}, Ontotext Refine is also packaged and distributed as a Docker image.

Quick Start

The following \textit{Docker Compose} file can be used to quickly get Refine up and running.

```
version: "3"

services:
  refine:
    image: ontotext/refine
    restart: unless-stopped
    environment:
      SERVER_HOST: localhost
    ports:
      - "7333:7333"
    volumes:
      - refine_data:/opt/ontorefine/data

volumes:
  refine_data:
```

After saving the Docker Compose file, start it by executing the next shell command in the same folder as the file:

```
docker-compose up -d
```

Refine is accessible by default at \url{http://localhost:7333/}.

\textbf{Note:} It is advisable to update the \texttt{SERVER\_HOST} environment variable to the actual hostname.
Refine Documentation, Release 1.2.0

Configuring Docker

You can pass any of the configurations parameters described below as environment variables in uppercase convention. For example, server.host should be written as SERVER_HOST.

Refine Configuration Parameters

You can assign configuration parameters in three ways:

• Edit the configuration file.
• Set corresponding environment variables. E.g. the property server.host should be written as SERVER_HOST.
• Add command-line options to the startup script. You can also pass JVM options that way. For example, to change the config file location, Refine port number, and set Java max heap, use:

```bash
ontorefine -Dontorefine.config.path=<your-config-file> -Dserver.port=<your-port> -Xmx4g
```

If the same parameter is set in multiple locations, the following order of precedence applies: config file < environment variables < command-line arguments.

Refine takes the following configuration parameters:

• ontorefine.config.path: Location of Refine config file. Created on first startup of Native Refine. Can be changed using the Java command-line property ontorefine.config.path or environment variable ONTOREFINE_CONFIG_PATH, but not within the config file itself. Defaults:
  - On Mac: {user.home}/.ontorefine/ontorefine.properties
  - On Windows: {user.home}\AppData\Roaming\OntoRefine\ontorefine.properties
  - On Linux: {user.home}/.ontorefine/ontorefine.properties
• ontorefine.data: Location of Refine data (project files and workspace.json). Defaults:
  - On Mac: {user.home}/.ontorefine
  - On Windows: {user.home}\AppData\Roaming\OntoRefine\refine
  - On Linux: {user.home}/.ontorefine
• ontorefine.dist: Refine installation directory. Can be set during installation of Native Refine. Should not be edited, unless Refine is moved manually. Defaults:
  - On Mac: /Applications/Ontotext Refine.app/Contents/app
  - On Windows: {user.home}\AppData\Local\Ontotext Refine\app
  - On Linux: /opt/ontorefine/lib/app/
• server.host: server on which Refine runs. Useful for CORS configurations. Default: your machine’s hostname
• server.port: port on which the Refine server responds. Default: 7333
• graphdb.address: URL address of the GraphDB instance that Refine asks for namespaces and class/property autocompletion. Default in the Setup menu of the web app: http://localhost:7200
• graphdb.repository: ID of the GraphDB repository that Refine works with. Default in the Setup menu of the web app: refine
• graphdb.auth.method: Authentication method used for GraphDB Access Control. Case-insensitive. Default: NONE. Possible settings:
– NONE: used only when GraphDB security is OFF. This should only be used on a local network not connected to the internet.

– BASIC: the username and password are sent in a header as plain text (see Basic authentication). Sending the password as plain text is insecure, so it is important to use Encryption in Transit (i.e. https) to improve security.
  * `graphdb.auth.basic.username`: username for Basic authentication.
  * `graphdb.auth.basic.credentials`: password for Basic authentication.

– GDB: token-based authentication, also used by GraphDB Workbench (see GDB authentication). Refine uses the GraphDB login API method to obtain a token: this happens on Refine startup, and whenever the token expires. Token expiration is controlled by the `graphdb.auth.token.validity` parameter in GraphDB (default: 30d). Refine uses the following parameters to perform GraphDB login:
  * `graphdb.auth.gdb.username`: username for GDB authentication.
  * `graphdb.auth.gdb.credentials`: password for GDB authentication.

Important: You can also connect a GraphDB instance using the visual dialog in Refine’s “Setup” section.

---

**GraphDB Setup**

![GraphDB Setup](image)

**Remote GraphDB Configurations**

- **Address**: http://localhost:9200
- **Repository ID**: refine
- **Authentication**: None
  - **Username**: nikola
  - **Password**: ********

[Save Configurations]
JVM Configuration

You can use the following environment variables to control Java startup parameters:

- **ONTOREFINE_JAVA_OPTS**: Sets additional Java parameters (-D or -X)
- **ONTOREFINE_JAVA_32BIT**: If set to true, configures Refine to run on 32bit Java
- **ONTOREFINE_MIN_MEM**: Sets the Java minimum heap size (-Xms option).
- **ONTOREFINE_MAX_MEM**: Sets the Java maximum heap size (-Xmx option).
- **ONTOREFINE_HEAP_SIZE**: Sets the Java minimum and maximum heap size (-Xms and -Xmx option).
- **ONTOREFINE_HEAP_NEWSIZE**: Sets the initial and maximum heap size for the young generation (-Xmn option).
- **ONTOREFINE_GC_LOG**: If set to true, enables the logging of Java garbage collection.
- **ONTOREFINE_GC_LOG_FILE**: Specifies a custom file for logging of Java garbage collection.

Other Java options are defined at the end of the file `{ontorefine.dist}/Ontotext Refine.cfg`.

Each `java-options=` line provides a single argument passed to the JVM when it starts (`-cp` means "class path").

Here are the options with their default settings on Windows:

```
[JavaOptions]
java-options=-Djpackage.app-version=1.0
java-options=-cp
java-options=${APPDIR}\ontorefine-native-app.jar;${APPDIR}\lib\*
java-options=-Dfile.encoding=UTF-8
java-options=-Dontorefine.dist=${APPDIR}
java-options=-Dadd-exports
java-options=jdk.management.agent/jdk.internal.agent=ALL-UNNAMED
java-options=-Dadd-opens
java-options=java.base/java.lang=ALL-UNNAMED

java-options=java.base/java.lang=ALL-UNNAMED
```

It is recommended not to remove or change any of the existing options provided with the installation. You can add your own options at the end.

For example, if you want to run Refine with 8 Gb of maximum heap memory, you can set the following option:

```
java-options=-Xmx8g
```

The launcher spawns the Refine application within the same JVM that it operates in, so you cannot change JVM options from the command line.

### 2.2 Migrating Existing Projects from GraphDB 9

Migrating projects from existing GraphDB instances to Refine is possible in bulk or manually.
2.2.1 Manual Migration

You can use manual migration for a small number of projects:

- Export the projects one by one from GraphDB 9 using the “Export project” function
- Import the projects one by one to Refine 1.0 using the “Import project” function

2.2.2 Bulk Migration

Bulk Migration migrates all projects from a Refine workspace at once.

- Make sure that the old OntoRefine (part of GraphDB 9) is not being used
- There are several ways in which you can transfer the workspace (set of projects):
  1. Set the `{ontorefine.data}` configuration parameter (of the new Refine) to point to the old data folder
  2. Replace the entire contents of the new (Refine) data folder with the contents of the old data folder
  3. Move project subfolders from the old data folder to the new data folder. (If a project ID already exists in the new data folder, leave it alone). Then edit the `projectIDs` array in `workspace.json` to list all projects.
    - This alternative is for advanced users only
    - Make sure you keep a backup of the old and new workspace folders
- Restart Refine in order to reload the projects
Ontotext Refine is built on top of OpenRefine. This is just a quick overview of the procedure for loading data into the tool. For the full set of capabilities of OpenRefine, please refer to its user manual.

### 3.1 Creating a Project

1. Start Refine.

All data files in Refine are organized as projects. One project can have more than one data file.

The *Create Project* action area consists of three tabs corresponding to the source of data. You can upload a file from your computer, specify the URL of a publicly accessible data, paste data from the clipboard or use a database.

1. Click *Create Project ▶ Get data from*.
2. Select one or more files to upload:
   - from your computer
   - from web addresses (URLs)
   - from clipboard
3. Click Next.

4. (Optional) Change the table configurations and update the preview.

   With the first opening of the file, Refine tries to recognize the encoding of the text file and all delimiters.

5. Click Create Project.
3.2 Importing a project

To import an already existing Refine project:

1. Go to Import Project.
2. Select a file (.tar or .tar.gz)
3. Import it.

3.3 Opening a project

Once the project is created:

1. Go to Open Project.
2. Click the one you want to work on.
3. (Optional) You can also delete your project if you want to.

The result of each of these actions is a table similar to that of an Excel or a Google sheet:
3.4 Saving and Exporting a Project

A refine Project consists of the data being manipulated and the metadata, containing information such as all the configurations, the history of operations, the mappings, etc...

Actions on the Open Refine project are saved automatically as part of the project metadata.

Actions in extensions, such as the RDF mapping tools, need to be saved manually. When saved they become part of the project metadata.

Exporting the project is done from the Export ▶ OpenRefine Project to a File.

3.5 Exporting the Project Configuration

The Project Configurations consist of:

- **The import options**: set of instructions on how to interpret the input file
- **Open Refine operations**: all the individual operations, performed on the data
- **RDF Mappings**, as defined in the RDF mapping extension

The user can export them using the Export ▶ Export project configurations menu item.

The resulting file can then be used to apply the same transformations on identically structured data using the Create, Apply Operations and Transform commands of the Ontotext Refine CLI.

3.6 Setting a Project Alias

A Project Alias is a user-defined identifier for a Open Refine project. The purpose of the project alias is to provide a means of accessing a Refine project’s virtual SPARQL endpoint, which can be controlled by the user and does not rely on the automatically generated project ID. See also Data Integration Using the Virtual SPARQL Endpoint.

Examples of such cases can be:

- setups involving multiple instances of Refine (such as a development / production split)
- setups in which identical transformations are applied on different sets of input data

A project alias can be set:

- from the GUI, using the field in the top row
- from the CLI, using the Update Aliases command

- A project can have many aliases.
- An alias can be any combination of alphanumeric characters and _ and should be up to 16 characters in length
- Aliases are unique in the context of all projects (i.e two or more projects can not share an alias).
- Aliases are case-sensitive - Alias and alias are treated as two different aliases
- When a project has an alias set the URL for accessing the SPARQL endpoint becomes BASE URL/repositories/ontorefine:PROJECT_ALIAS

  e.g. <http://localhost:7333/repositories/ontorefine:my_project> will access the SPARQL endpoint of a project with a my_project alias on a Refine instance running on http://localhost:7333

If many aliases are set, all of them can be used to access the endpoint. In such cases the Refine RDF Mapper will consider by default one of the aliases, when generating the queries with SERVICE clause.
Ontotext Refine is designed to facilitate the production of RDF from tabular data. Depending on the particular use case and requirements, the user can choose either the visual RDF mapping interface or directly write your mapping as a SPARQL query. Ontotext Refine also exposes a virtual SPARQL endpoint on top of the tabular data, which can be used in more complex data integration scenarios.

4.1 Connecting to a GraphDB instance

The user can connect a GraphDB instance using the Setup page in the Refine Menu or by setting the corresponding Refine Configuration Parameters.

Connecting to a GraphDB instance is not required in order to produce RDF. However it will further facilitate your work by allowing:

- Auto-completion of classes and properties from an existing ontology
- Access to prefixes
- Easy access to the virtual SPARQL endpoint directly from the GraphDB workbench

Important: If using GraphDB to autocomplete classes and properties make sure that autocomplete is enabled

4.2 Using The Visual RDF Mapper

The Visual RDF Mapper allows mapping the tabular data progressively and interactively, one triple at a time.

4.2.1 The Mapper User interface

The headers of all columns in the tabular data are displayed as boxes that you can drag and drop into a mapping cell to configure them.

Following are the BASE and PREFIX definitions. New prefixes can be added using SPARQL syntax. Prefixes can be edited and copied.
The Mapping Table contains the actual mapping. Each row in the table represents an RDF triple constructed from the tabular data. If two (or more) triples have the same subject, they will be displayed as a triple with one subject and two (or more) predicates. Analogically, if two (or more) triples have the same subject + predicate, but different objects, they will be displayed as a triple with one subject + predicate and multiple different objects.

The mapping is applied consecutively to each row in the input table.

**Note:** The table is modeled after the Turtle language syntax. If you look closely you will see the familiar use of the . , ; and , separators at the end of each row.

### 4.2.2 Creating a mapping

Creating a new mapping is done one step at a time by configuring the values at the **Subject**, **Predicate** and **Object** positions of each triple.

For each position in the mapping table the user defines a value mapping, i.e. he describes how to create the RDF value based on the input tabular data. In order to determine the value, the user should specify the **value source**, the **value type** and the **value transformation**.

**The Value Source**

This is the source of the value in the associated position, it can be one of the following:

- **Column**: the value of the cell in the mentioned column
- **Constant**: a constant value
- **Record ID**: The record identifier from the OpenRefine project
- **Row Index**: The index of the row
- **GREL**: a GREL expression
Tip: You can also refer to a column directly from a cell in the mapping table by typing @ followed by the name of the column.

The Value Type

The value type defines the type of RDF value that will be produced by the value mapping. The possible types are:

- **Resource (abstract):** An RDF resource. This is an abstract type that specifies the common features of IRIs and blank nodes. A resource value type may have type mappings and property mappings on its own.

- **IRI:** An RDF IRI. The transformed value is the IRI with illegal characters escaped automatically. This is a subtype of the abstract Resource value type.

- **Blank node based on value:** (value Bnode) An RDF blank node. The transformed value is used to calculate a reasonable blank node identifier such that identical transformed values produce the same blank node. This is a subtype of the abstract Resource value type.

- **Unique blank node:** (unique Bnode) An RDF blank node. The transformed value is ignored and a unique blank node is created every time. This is a subtype of the abstract Resource value type.

- **Any literal (abstract):** Any kind of RDF literal (plain, language, or datatype). This is an abstract type that unifies all literal value types.

- **Literal:** An RDF plain literal. The transformed value is the literal’s label.

- **Literal with a language:** An RDF literal with a language. The transformed value is the literal’s label. The language is a simple literal value mapping, i.e., identical to a value mapping with type literal.

- **Literal with a datatype:** An RDF literal with a datatype. The transformed value is the literal’s label. The datatype is a simple IRI value mapping.

This window shows how to produce a xsd:gYear typed literal from the values of the startYear column.
The Value Transformation

Each value mapping may have an optional transformation, applied to the data received from the value source, before the value is created.

Refine supports two types of transformations:

- **GREL**: A GREL expression defines the transformation
- **Prefixing**: A known prefix is applied to the value and the result is given the IRI data type

This window shows a GREL expression being used to construct a URL from the values of the `tconst` column.

Note the instant preview of the result of the GREL expression.
4.2.3 Prefixes

The mapping tree contains a set of prefixes that are used in the cell configuration. They are defined in the prefix area and can be of three types:

- Default prefixes from commonly used RDF schemas, such as foaf, geo, rdf, rdfs, skos, xsd. You can select entities from these schemas without importing them in your repository.
- Prefixes that you select from the GraphDB repository,
- Prefixes that you create yourself.

**Tip:** It is possible to extend prefixes when using them in the visual mapper. Entering movie:episode/ will extend the predefined movie: prefix with episode/.

4.2.4 Preview Mode

Besides the default Configuration mode, the RDF mapper can be switched to Preview mode. This mode shows the RDF resulting from mapping the first row of the table. The URIs are clickable and can be used to validate if they are well constructed.
4.2.5 Nesting Triples and Blank Nodes

It is possible to nest triples directly from the Visual RDF Mapper.

Clicking on the arrow symbol of a the cell in the object position will start a nested set of triples, where that resource is implicitly in the subject position. This is indicated by the green frame around the nested triples.

The following mapping:

Produces the following RDF:

```turtle
movie:tt0093296 mdbo:starring [ mdbo:actor actor:nm0141056; mdbo:ordering 3; mdbo:character "Heinrich", "Herman", "Hans"
]
```

from the following source:

```
<table>
<thead>
<tr>
<th>tconst</th>
<th>ordering</th>
<th>nconst</th>
<th>category</th>
<th>job</th>
<th>characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt0093296</td>
<td>3</td>
<td>nm0141056</td>
<td>actor</td>
<td>[&quot;Heinrich&quot;,&quot;Herman&quot;,&quot;Hans&quot;]</td>
<td></td>
</tr>
</tbody>
</table>
```

**Note:** The transformation above is not trivial.

Note how we used a GREL forEach to parse the JSON string and and produce three triples with the mdbo:character predicate.
4.2.6 Reusing a Mapping

Once created, a mapping can be saved using the Save button.

It will be saved as part of the overall project. A mapping can be exported as a JSON file and saved using the Download JSON button. Respectively such a file can be restored with the Upload JSON button.

The user can also reuse a mapping by:

- Convert it to SPARQL open it in a GraphDB query editor (see below)
- Convert it to SPARQL from the Refine SPARQL editor (see below)

4.3 Using the SPARQL Query Editor

The SPARQL Query Editor allows directly creating and editing mappings as SPARQL queries.

The input tabular data is read one row at a time and the value in each cell is bound as a variable, named after the column header.

The user is then free to manipulate the values with all the expressiveness that SPARQL allows.

The output RDF is shaped, using either a CONSTRUCT query or a INSERT query, from a remote endpoint, using SPARQL federation.
4.3.1 Mapping Syntax and Dedicated SPARQL Variables

The mapping query uses standard SPARQL syntax with a few special variables and statements.

The Mapping Variables

Values from the input table are mapped to variables, named after the corresponding columns, prefixed with a `c_`.

These variables are not readily available and they must be mentioned in a `BIND()` statement in order to be activated.

The values are always initially typed as strings and we recommend using the same `BIND()` statement to handle any data type conversions. If the desired datatype is string, the variable still needs to be bound as a variable with a different name.

Here is an example with the following conversions:
- the contents of the `startYear` is converted to a literal with a `xsd:gYear` data type,
- the `runtimeMinutes` column is converted to an integer,
- the `primaryTitle` column is not converted and remains a string.

Tip: Click on the name of the columns on top of the window!

The generic `BIND()` statement corresponding to that column will be generated in the `WHERE` clause of the query.

The Row and Record index

Two more variables are reserved by the SPARQL mapping. They correspond to OpenRefine project metadata:
- `?row_index` contains the index of the row being processed
- `?record_id` contains the id of the record being processed (if in record mode)

The `mapper:grel` Magic Predicate

GREL expressions can be used directly in the SPARQL query, using the `mapper:grel` magic predicate. The subject part of the triple contains an expression with two arguments, the `?rowIndex` and a string corresponding to the GREL expression itself. The variable in the object position contains the result.

Hint: A lot of the tasks for which we can use GREL can also be achieved by SPARQL 1.1 string functions, with much less syntactic complexity.
4.3.2 Making mappings using SPARQL

While it is possible to just start from scratch and write a query, Ontotext Refine has several features for bootstrapping this process.

Generating such mapping queries can be done either from the Visual RDF Mapper or from a predefined template.

Generate a Query from the Visual RDF Mapper

If the user has defined a mapping by using the Visual RDF Mapper, the Generate Query ▶ From Mapping menu item will translate the mapping to a SPARQL CONSTRUCT query.

This SPARQL query:

```sparql
CONSTRUCT {  
  ?s1 a mdbo:Movie ;  
  mdbo:title ?o_title ;  
  mdbo:year ?o_year .  
}  
WHERE {  
  BIND(IRI(mapper:encode_iri(movie:, ?c_tconst)) as ?s1)  
  BIND(STR(?c_primaryTitle) as ?o_title)  
  BIND(STRDT(?c_startYear, xsd:gYear) as ?o_year)  
}
```

Has been automatically generated from the following visual mapping:

```
Important: Any modifications to the query are not reflected back to the Visual RDF Mapper.

Generate a Query from Template

If a user wants to start directly working with SPARQL, the Generate Query ▶ Standard Template option is the easiest way to start.

This feature will produce a SELECT query with all the BIND() statements, corresponding to the columns of the tabular data.

The recommended workflow in this scenario is the following:

- First the user chooses which columns he needs and removes the unneeded BIND() statements
- Second, he fixes any data type issues, directly in the BIND() statements using STRDT().
- He finally crafts the URIs so they fit the URI format of his project.

This is an iterative process the user can execute the query many times in order to inspect the results.

Once satisfied, the user changes the SELECT to a CONSTRUCT query and defines the shape of the output RDF graph.
The result in Turtle format can then be downloaded using the Download □ Result menu item.

**Tip:** The SPARQL Mapper can be a powerful data exploration tool! You can use it to check for inconsistencies, to count occurrences, and in general to get a better understanding of the data you are transforming. Also remember that you can use an additional BIND( ) or FILTER( ) statements, to fix a certain value and constrain result set to a small subset of the data, centered around a (potentially problematic) value.

Alternatively if the project has too many columns, the individual BIND( ) statements can be generated one by one by clicking on the names of the columns in the top band.

**Important:** Values from the tabular data always come typed as strings. It is up to the user to convert them to the desired data type using the STRDT( ) SPARQL function.

### Saving the Mapping Queries

All the mapping queries in the **SPARQL Query Editor** are saved as part of the Refine project. This is done after clicking on the **Save** button but **is only effective for queries after they have been executed**.

**Warning:** Modifications in a query are not saved if the query has not been executed after having been modified.

The queries can also be saved using the **Download □ Query** menu item.

**Tip:** The tabs can be renamed by double-clicking on their names.

### 4.4 Data Integration Using the Virtual SPARQL Endpoint

Thanks to SPARQL federation, advanced users can produce RDF using Ontotext Refine by using the SPARQL endpoint directly from a GraphDB instance.

This is the recommended method for integrating data using Ontotext Refine for a number of reasons:

- A user can use SPARQL INSERT to directly write triples in the GraphDB repository
- Federated queries allow joining values from the Refine project and the existing data in the graph. This allows for much finer control on the shape of the resulting RDF graph.
- No intermediary files are created
- The user can specify the context using a SPARQL GRAPH clause and can even define it dynamically from values in the data.
4.4.1 Accessing the Virtual Endpoint

Ontotext Refine exposes a SPARQL endpoint for each project.

The URL of the endpoint corresponds to the Refine base URL, followed by `repositories/ontorefine`, followed by the identifier of the project.

If a Project Alias is set it will be used in place of the project identifier to form the virtual endpoint URL.

Aliases are useful when the mapping queries are stored externally and need to be persistent, because they will be used on newly created projects.

In order to run the federated queries, Refine can generate the queries with the corresponding SERVICE clause, wrapping the statements in the WHERE clause. This is done by choosing the corresponding menu items in the Generate query menu:

The Open in GraphDB button also inserts the SERVICE clause in the current query before opening it in the associated GraphDB instance.

Note: The URL of the virtual SPARQL endpoint might need to be changed, to fit the particular user’s architecture. Such are cases where Refine runs inside a docker container or needs to be accessed from another machine.

This window shows a INSERT query, ready to write triples in a GraphDB repository.
5.1 Overview

The main Refine functionalities can be used via the Refine command line interface (CLI), which uses a REST API for executing Refine operations without having to interact with its UI. This is quite useful for automating data pipeline steps: cleaning up, transforming, enriching, storing and exposing search operations over various datasets.

5.1.1 Source Code & Main Dependencies

The CLI is developed as an open source project and it is available at ontorefine-cli GitHub. The main purpose of the CLI is to expose the functionalities that ontorefine-client provides through command interface. The Ontotext Refine Client (ORC) is the another open source project that is maintained by Ontotext and provides a convenient API for communication with OR.

Picocli is the other main dependency that is used in the CLI. It allows the development of the rich command line applications that can run on and off the JVM. Each CLI command is a combination of the ORC API and Picocli, where the latter provides the means to pass the required arguments for the different operations of the ORC.

5.1.2 Distribution & Usage

Currently the CLI is distributed as part of the Ontotext Refine (OR) standard toolbox. As such it is available in all distributions that OR has, which includes the OS native and the Platform Independent distributions. Even though the CLI is distributed with specific OR, it has the ability to work with any compatible remote instance of OR. This is enabled by the --url argument that needs to be provided for each command.

For convenience, there are invocation scripts for the different operating systems, allowing the execution of commands to be simple and easy. These scripts are placed in the bin directory of the OR distribution:

- ontorefine-cli.sh (Unix)
- ontorefine-cli.cmd (Windows)

The execution of CLI commands is done by invoking the ontorefine-cli script via terminal and providing all of the required arguments for the specified command.
5.2 Commands

This section provides information for all currently supported commands. For each command are listed:

- A brief description of the command and what it does
- An example how to execute it
- A list of the arguments that it accepts

For consistency in all examples, we will use the same dataset containing information about restaurants in the Netherlands available at: Netherlands_restaurants.csv. Also we assume that the Ontotext Refine instance was started locally, using the default port, meaning that the instance address is http://localhost:7333.

5.2.1 Create

Creates a Refine project using the provided dataset.

The command uploads the dataset to Ontotext Refine, which creates an project with a unique identifier. The project identifier is returned as a response from the command.

```bash
# Getting help for the command
ontorefine-cli create --help

# Output
Usage:
ontorefine-cli create [-hV] [-c <configurations>] [-f <format>] [-n <name>] -u <url> [-a <aliases> [, <aliases> ...]]... FILE

Description:
Creates a new project from a file.

Parameters:
  FILE The file that will be used to create the project. It should be a full name with one of the supported extensions (csv, tsv, json, xls, xlsx, etc.).

Options:
  -u, --url <url> The URL of the Ontotext Refine instance to connect to, e.g. http://localhost:7333.
  -n, --name <name> A name for the Refine project. If not provided, the file name will be used.
  -f, --format <format> The format of the provided file. The default format is `csv`. Except `csv`, all other formats are in experimental state. The allowed values are: csv, tsv, excel, json, xml
  -c, --configurations <configurations> File containing configurations for the importing process of the dataset. It includes information how to parse the input data so that it can be represented in tabular form and additioanl options related to the project creation.
  -a, --aliases <aliases> [, <aliases> ...] Aliases for the project. The argument accepts multiple comma separated values.
  -h, --help Show this help message and exit.
  -V, --version Print version information and exit.
```

Example
5.2.2 Delete

Deletes a specific project from the Ontotext Refine workspace.

The command uses the provided identifier to remove the project and its data. The result from the command is a message with the status from the execution.

### Getting help for the command

```
ontorefine-cli delete --help
```

### Output

```
Usage:
ontorefine-cli delete [-hV] -u <url> PROJECT

Description:
Deletes a project from Ontotext Refine.

Parameters:
    PROJECT  The identifier of the project that should be deleted.

Options:
    -u, --url <url>  The URL of the Ontotext Refine instance to connect to, e.g. http://localhost:7333.
    -h, --help       Show this help message and exit.
    -V, --version    Print version information and exit.
```

**Example**

```
# Command
ontorefine-cli delete 2121442084816 -u http://localhost:7333

# Output
Successfully deleted project with identifier: 2121442084816
```

5.2.3 Export

Exports the data of a given project in CSV format.

The command extracts the data of the project and transforms it to CSV format. The result of the command is the project data in the requested format.

### Getting help for the command

```
ontorefine-cli export --help
```

### Output

```
Usage:
ontorefine-cli export [-hV] -u <url> PROJECT FORMAT

Description:
```
Exports the data of a project in CSV format.

Parameters:
- **PROJECT** The identifier of the project to export.
- **FORMAT** The output format of the export (only csv at the moment).

Options:
- `-u`, `--url <url>` The URL of the Ontotext Refine instance to connect to, e.g. `http://localhost:7333`.
- `-h`, `--help` Show this help message and exit.
- `-V`, `--version` Print version information and exit.

Example

```bash
# Command
ontorefine-cli export 2121442084816 csv -u http://localhost:7333
```

# Output

Trcid,Title,Shortdescription,Longdescription,Calendarsummary,TitleEN,
 ShortdescriptionEN,LongdescriptionEN,CalendarsummaryEN,Types,Ids,Locatienaaam,
 City,Adres,Zipcode,Latitude,Longitude,Urls,Media,Thumbnail,Datepattern_startdate,
 Datepattern_enddate,Single dates,Types,Ids,Locatienaaam,
 City,Adres,Zipcode,Longitude
669d7d82-8962-4e88-b2e1-7b8706633aa0,Smits Noord-Zuid Hollandsch Koffiehuis,Het u
 onderhield met Amsterdam naar het noorden van de provincie en is in 1919 gebouwd.
 Nu is er een restaurant en een koffiebar. Ook is hier een informatiekantoor van
 Amsterdam Marketing gehuisvest.,,,Smits Noord-Zuid Hollandsch Koffiehuis,"The
 Smits Koffiehuis dates back to 1919. This charming building served as the
 departure and arrival point for a steam tram that once connected Amsterdam to
 the northern parts of the Noord Holland province. In addition to the restaurant
 and café, this beautiful landmark in front of Central Station also houses the
 Tourist Information Office and a GVB (public transport) office. ",,,3.1.1,,
 smitskoffiehuis.nl,https://media.iamsterdam.com/ndtrc/Images/20101122/ec8faec5-
 5cd5-43d6-b0fa-eb0dab65e278.jpg,https://media.iamsterdam.com/ndtrc/Images/
 20101122/ec8faec5-5cd5-43d6-b0fa-eb0dab65e278.jpg,,,,2015-10-09 14:04:44,
 # And more ...

5.2.4 Extract Operations

Extracts all operations applied to a project.

The command makes a request to Ontotext Refine to extract the history of the operations which were applied to the data. The result from the command is a JSON document containing the applied operations or empty array, if no operations are applied to the specified project.

# Getting help for the command
ontorefine-cli extract --help

# Output

Usage:

ontorefine-cli extract [-hV] [-m <mode>] -u <url> PROJECT

Description:

Extracts specific project configuration in JSON format.

Parameters:
- **PROJECT** The project whose configurations to extract.
Options:
- `--url <url>` The URL of the Ontotext Refine instance to connect to, e.g. http://localhost:7333.
- `--mode <mode>` Controls which project configuration to be extracted. The default is 'operations'. The allowed values are: operations, import-options, full.
- `--help` Show this help message and exit.
- `--version` Print version information and exit.

Example

```bash
# Command
ontorefine-cli extract 2121442084816 -u http://localhost:7333

# Output
[
  {
    "op": "core/text-transform",
    "engineConfig": {
      "facets": [],
      "mode": "row-based"
    },
    "columnName": "City",
    "expression": "value.toTitlecase()",
    "onError": "keep-original",
    "repeat": false,
    "repeatCount": 10,
    "description": "Text transform on cells in column City using expression value. value.toTitlecase()"
  }
]
```

### 5.2.5 Apply Operations

Applies operations to a specified project.

The command uses the provided JSON document with operations and/or project configurations and to applies them to the project. The result of the command is a message with the status of the execution. See also Exporting the Project Configuration for more information.

**Note:** If the file includes import options they will be ignored.

```bash
# Getting help for the command
ontorefine-cli apply --help

# Output
Usage:
ontorefine-cli apply [-hV] -u <url> OPERATIONS PROJECT

Description:
Applies transformation operations to a project.

Parameters:
  OPERATIONS  The file with the operations that should be applied to the project. The file should be a JSON file.
  PROJECT     The identifier of the project to which the transformation
```

(continues on next page)
operations will be applied.

Options:
- **-u, --url <url>** The URL of the Ontotext Refine instance to connect to, e.g. http://localhost:7333.
- **-h, --help** Show this help message and exit.
- **-V, --version** Print version information and exit.

### Example

To obtain a operations.json file, you can use the extract command as shown above or extract the operations from the Ontotext Refine Web.

```bash
# Command
ontorefine-cli apply operations.json 2121442084816 -u http://localhost:7333
# Output
The transformations were successfully applied to project: 2121442084816
```

### 5.2.6 Register Reconciliation Service

Registers an additional service for reconciliation that can be used in the Ontotext Refine web interface. The command registers the new service address by executing an request to Ontotext Refine REST API. The result of the command is a message with the status of the request execution.

```bash
# Getting help for the command
ontorefine-cli register-service --help
# Output
Usage:
ontorefine-cli register-service [-hV] -u <url> SERVICE
Description:
Registers an additional reconciliation service.
Parameters:
SERVICE The URL of the additional service that should be registered.
Options:
- **-u, --url <url>** The URL of the Ontotext Refine instance to connect to, e.g. http://localhost:7333.
- **-h, --help** Show this help message and exit.
- **-V, --version** Print version information and exit.
```

### Example

```bash
# Command
# Output
Successfully registered additional reconciliation service: 2121442084816
```
5.2.7 RDF Export

Exports the data of a given project in RDF format.

The command extracts and converts the data of specified project into RDF format using Ontotext Refine’s internal SPARQL engine. It supports two mechanisms for conversions. One via a JSON mapping and the other via SPARQL CONSTRUCT query. The query takes precedence if both arguments are provided. As fallback, if neither mapping, nor SPARQL is provided, the command will try to retrieve the mappings from the operation history for the project. The result of the command is the project data in specific RDF format.

# Getting help for the command
ontorefine-cli rdf --help

# Output
Usage:
ontorefine-cli rdf [-hV] [-f <format>] [-m <mapping>] [-q <sparql>] -u <url> -p PROJECT

Description:
Exports the data of a project to RDF format.

Parameters:
   PROJECT The project whose data to convert to RDF.

Options:
   -u, --url <url> The URL of the Ontotext Refine instance to connect to, e.g. http://localhost:7333.
   -q, --sparql <sparql> A file containing SPARQL CONSTRUCT query to be used for RDF conversion.
   -m, --mapping <mapping> The mapping that will be used for the RDF conversion. The file should contain JSON configuration. If not provided the process will try to retrieve it from the project configurations.
   -f, --format <format> Controls the format of the result. The default format is 'turtle'. The allowed values are: rdfxml, ntriples, turtle, turtlestar, trix, trig, trigstar, binary, nquads, jsonld, rdfjson
   -h, --help Show this help message and exit.
   -V, --version Print version information and exit.

Example
Mapping JSON for the example: restaurants-mapping.json

# Command
ontorefine-cli rdf 2121442084816 -m restaurants-mapping.json -u http://localhost:7333

# Output
@base <http://example/base/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://schema.org/> .
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix amsterdam: <https://data/amsterdam/nl/resource/> .
@prefix sf: <http://www.opengis.net/ont/sf#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
<https://data/amsterdam/nl/resource/restaurant/669d7d82-8962-4e88-b2e1-7b8706633aa0>
a schema:Restaurant;
  schema:description "Het Smits Koffiehuis ontleent haar ontstaan aan de stoomtram, die de verbinding onderhield met Amsterdam naar het noorden van de provincie en is in 1919 gebouwd. Nu is er een restaurant en een koffiebar. Ook is hier een informatiekantoor van Amsterdam Marketing gehuisvest."
  schema:latitude "0"^^xsd:float;
  amsterdam:zipcode "1012 AB";
  schema:image <https://media.iamsterdam.com/ndtrc/Images/20101122/ec8faec5-5cd5-43d6-b0fa-eb0dab65e278.jpg>;
  geo:hasGeometry <https://data/amsterdam/nl/resource/geometry/669d7d82-8962-4e88-b2e1-7b8706633aa0>;
  amsterdam:uniqueLocation :node1gam2kjl2x1;
  amsterdam:valuelocation _:node1gam2kjl2x1;
  amsterdam:address "Stationsplein 10".
</https://data/amsterdam/nl/resource/geometry/669d7d82-8962-4e88-b2e1-7b8706633aa0>
  a sf:Point;
  _:node1gam2kjl2x1 amsterdam:address "Stationsplein 10".
  _:669d7d82-8962-4e88-b2e1-7b8706633aa0 amsterdam:city "Amsterdam".
# And more ...

## 5.2.8 Transform

Transforms a dataset into another specific format.

The command represents a composition of several other commands in order to allow complete transformation pipeline for processing of datasets. The phases of the command are:

- create project
- apply operations, if there are any
- export the data in the specified format using the provided mapping or SPARQL query
- delete the project

At the moment the command supports only transformation of CSV to RDF, but it will be gradually extended with more options.

```bash
# Getting help for the command
ontofine-cli transform --help
```

**Output**

**Usage:**

```bash
ontofine-cli transform [-hV] [--[no-]clean] [-c <configurations>] [-f <format>] [-q <sparql>] [-r <result>] -u <url> [-a <aliases>[,<aliases>...]]... FILE
```

**Description:**

Transforms given dataset into different data format.

**Parameters:**

- **FILE**
  The file containing the data that should be transformed. It should be a full name with one of the supported extensions: (csv, tsv, json, xls,
Options:

- `u, --url <url>` The URL of the Ontotext Refine instance to connect to, e.g. http://localhost:7333.

- `f, --format <format>` The format of the provided file. The default format is `csv`. Except `csv`, all other formats are in experimental state. The allowed values are: csv, tsv, excel, json, xml

- `c, --configurations <configurations>` A file with the configurations that should be used for project creation. Ideally it should contain the import options and the operations history of the project, but it is also allowed for one of the configurations to be present. The mapping for the RDFization of the dataset is stored as operation to the history. The file should contain JSON document.

- `q, --sparql <sparql>` A file containing SPARQL CONSTRUCT query to be used for RDFization of the provided dataset.

- `r, --result <result>` Controls the output format of the result. The default format is 'turtle'. The allowed values are: rdfxml, ntriples, turtle, turtletstar, trix, trig, trigstar, binary, quads, jsonld, rdfjson

- `a, --aliases <aliases> [...]` Aliases for the project. The argument accepts multiple comma separated values.

- `--[no-]clean` Controls the cleaning of the project after the operation execution. When enabled the clean up will be executed regardless of the success of the transformation. By default the cleaning is enabled.

- `h, --help` Show this help message and exit.

- `V, --version` Print version information and exit.

Example

Example CONSTRUCT query: construct.sparql

```bash
# Command
ontorefine-cli transform "Netherlands_restaurants.csv" -c operations.json -q construct.sparql -r turtle -u http://localhost:7333
```

# Output

```sparql
@base <http://example/base/>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix schema: <http://schema.org/>.
@prefix geo: <http://www.opengis.net/ont/geosparql/>.
@prefix amsterdam: <https://data/amsterdam/nl/resource/>.
@prefix sf: <http://www.opengis.net/ont/sf#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

<https://data/amsterdam/nl/resource/restaurant/669d7d82-8962-4e88-b2e1-7b870663aa0>
a schema:Restaurant;
  schema:description "Het Smits Koffiehuis ontleent haar ontstaan aan de stoomtram, die de verbinding onderhield met Amsterdam naar het noorden van de provincie en is in 1919 gebouwd. Nu is er een restaurant en een koffiebar. Ook is hier een informatiekantoor van Amsterdam Marketing gehuisvest."
  schema:latitude "0"^^xsd:float;
```
5.2.9 Refine Version

Retrieves the current version of the Ontotext Refine.

This command gets the version of Ontotext Refine via a REST API call. The result of the command is a message containing the requested information.

```
# Getting help for the command
ontorefine-cli refine-version --help

# Output
Usage:
ontorefine-cli refine-version [-hV] -u <url>

Description:
Retrieves the version of the Ontotext Refine instance.

Options:
- -u, --url <url>  The URL of the Ontotext Refine instance to connect to, e.g. http://localhost:7333.
- -h, --help         Show this help message and exit.
- -V, --version      Print version information and exit.
```

Example

```
# Command
ontorefine-cli refine-version -u http://localhost:7333

# Output
Name: OpenRefine  [1.2]
Full version:  [1.2]
Version:  1.2
Revision:  1.238 Chapter 5. Ontotext Refine CLI
5.2.10 Update Aliases

Updates project aliases.

This command adds or removes aliases for a given project. The command can add and remove values in a single invocation.

```sh
# Getting help for the command
ontorefine-cli update-aliases --help
```

```
# Output
Usage:
ontorefine-cli update-aliases [-hV] -u <url> [-a <add>,<add>...]... [-r <remove>,<remove>...]... PROJECT

Description:
Updates project aliases. The command can add and remove values in a single invocation.

Parameters:
- PROJECT The identifier of the project to which the transformation operations will be applied.

Options:
- -u, --url <url> The URL of the Ontotext Refine instance to connect to, e.g. http://localhost:7333.
- -a, --add <add> Aliases to add to the project. The argument accepts multiple comma separated values.
- -r, --remove <remove> Aliases to remove from the project. The argument accepts multiple comma separated values.
- -h, --help Show this help message and exit.
- -V, --version Print version information and exit.
```

**Example**

```sh
# Command
ontorefine-cli update-aliases -u http://localhost:7333/ -r "my_project" -a "our_project" 1681093299584
```

```
# Output
Successfully updated the aliases of project: 1681093299584
```

5.2.11 Help

Provides generic information about the supported commands in the CLI.

The command can be used in combination with another command to get information for the specified operation. The result from the command is a list of all available commands.

```sh
# Getting help
ontorefine-cli help
```

```
# Output
Usage: ontorefine-cli [-hV] [COMMAND]
  -h, --help Show this help message and exit.
  -V, --version Print version information and exit.

Commands:
  create Creates a new project from a file.
```

(continues on next page)
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete</td>
<td>Deletes a project from OntoRefine.</td>
</tr>
<tr>
<td>export</td>
<td>Exports the data of a project in CSV or JSON format.</td>
</tr>
<tr>
<td>apply</td>
<td>Applies transformation operations to a project.</td>
</tr>
<tr>
<td>register-service</td>
<td>Registers an additional reconciliation service.</td>
</tr>
<tr>
<td>rdf</td>
<td>Converts the data of a project to RDF format.</td>
</tr>
<tr>
<td>transform</td>
<td>Transforms given dataset into different data format.</td>
</tr>
<tr>
<td>refine-version</td>
<td>Retrieves the version of the Ontotext Refine instance.</td>
</tr>
<tr>
<td>update-aliases</td>
<td>Updates project aliases. The command can add and remove values in a single invocation.</td>
</tr>
<tr>
<td>help</td>
<td>Displays help information about the specified command</td>
</tr>
</tbody>
</table>
INSTALLING REFINE EXTENSIONS

6.1 Introduction

OpenRefine, the framework which Ontotext Refine is build upon, provides a mechanism for extending its functionalities. This extension mechanism is driven by the Butterfly framework. OpenRefine and, therefore, Ontotext Refine extensions, are Butterfly modules. However, no knowledge of Butterfly is necessary.

A set of default extensions are shipped together with the distribution, located in the extensions subdirectory. When you want to add your own extensions, you can place them anywhere, as long as you point Butterfly to it.

The extensions have to be placed on the same filesystem as Ontotext Refine.

6.2 Compatibility Notice

Many of the extensions which are developed for OpenRefine are out of date. You can check the current version which Ontotext Refine ships with from the “about” menu of the application.

| Warning: | Most outdated extensions will not run correctly, some would completely crash the application. |

The list of extensions which we have verified to run with Ontotext Refine are:

- database
- jython
- pc-axis
- phonetic
- wikidata
- rdf-transform

| Warning: | The gdata extension has a problem related to the way in which iframes are managed and has been disabled for this version of Ontotext Refine. We would like to incorporate it in the future. |
6.3 Developing New Extensions

You can use the tutorial provided by OpenRefine. Keep in mind that due to the specific server implementation of Ontotext Refine, extensions which add new popup windows will not function properly.

6.4 Extensions Directory

The easiest way to set up the extensions is to store them within the extensions directory. The path to the directory is relative to the application source, `lib/refine/extensions`.

Care should be taken when you are modifying the extensions directory as it is part of the application source.

6.5 Module Path Property

Setting up modules can be as easy as setting a new `butterfly.modules.path`. This will not override the default paths.

You can also set up multiple paths if your modules are not placed within the same location.

Alternatively, you can set up a list of directories in the same property: `-Dbutterfly.modules.path=modules,../../extensions,/path/to/your/extension`

6.6 Deployment-specific Configurations

6.6.1 Onto Refine Desktop

The Ontotext Refine native application has a few caveats, but the general approaches to setting the extensions directory still apply.

The extensions directory in the native app is `${ontorefine.dist}/app/lib/refine/extensions`.

You can set up the butterfly modules path in the `.cfg` file, at `${ontorefine.dist}/app/Ontotext Refine.cfg`. Add the line `java-options=-Dbutterfly.modules.path=modules,../../extensions,/path/to/your/extension` to the bottom of the file.

Alternatively, you can conveniently set up the `butterfly.modules.path` property in the UI, from the Settings window. If you have multiple module paths, you need to comma-separate them within the same property, as described above.

Finally, you can use the “Set Extensions and Restart” button. This opens your file explorer. You can select your extension folder from here. This sets the `butterfly.modules.path` property.

The drawback of this approach is that you can only set one path.
6.6.2 Standalone Distribution

The general approaches outlined above are also available for the standalone distribution.

The extensions directory in the native app is ${ontorefine.dist}/app/lib/refine/extensions.

You can use the –extensions startup argument to point towards the module paths, e.g. ./ontorefine.sh –extensions modules,../../extensions,/path/to/your/extension

You can set up the butterfly modules path within the environment variable ONTOREFINE_JAVA_OPTS.

Alternatively, you can directly pass the property by using the -D flag, e.g. ./ontorefine.sh -Dbutterfly.modules.path=modules,../../extensions,/path/to/your/extension.

6.6.3 Docker Distribution

When running from Docker, the same ways to set up the extension modules path as in the standalone application apply.

The only difference is that the modules need to be passed inside the container as a volume.

docker run ontotext/refine -v /path/to/your/extension:/container/internal/custom/extension -e ONTOREFINE_JAVA_OPTS=-Dbutterfly.modules.path=modules,../../extensions,/container/internal/custom/extension

6.7 Example Extension Setup

Here are the steps for setting up an extension with the native application:

• Set up your extension in a folder. The file structure needs to be the following:

```
extension-folder/
  extension-name/
    module/
```

• Set up the butterfly.modules.path to point towards the extension-folder.

• Restart Ontotext Refine to pick up your new extension.

**Note:** Butterfly searches for extensions recursively in sub-folders. If you add a folder with many sub-directories, the startup process will be slower. It’s also possible to get unexpected modules loaded. Therefore, we recommend a relatively flat structure.
What's in this document?

- Refine 1.2.2
- Refine 1.2.1
- Refine 1.2
  - Improvements and Bugfixes
  - Known Issues
- Refine 1.1
  - Improvements and Bugfixes
  - Known Issues
- Refine 1.0
  - Improvements
  - Known Issues

Refine release notes provide information about the features and improvements in each release, as well as various bug fixes. Refine's versioning scheme is based on semantic versioning. The full version is composed of three components:

major.minor.patch

e.g., 1.2.3 where the major version is 1, the minor version is 2, and the patch version is 3.

Note: Releases with the same major and minor versions do not contain any new features. Releases with different patch versions contain fixes for bugs discovered since the previous minor. New or significantly changed features are released with a higher major or minor version.

7.1 Refine 1.2.2

Released: 19 June 2023

The version contains overall update of all external dependencies in order to get the latest security patches and fixes. The update includes: libraries, docker base images, Java versions, etc.
7.2 Refine 1.2.1

Released: 24 February 2023
The version contains overall update of all external dependencies in order to get the latest security patches and fixes. The update includes: libraries, docker base images, Java versions, etc.

7.3 Refine 1.2

Released: 15 February 2023

Important: Refine 1.2:
- **New Feature**: SPARQL Query Editor.
- **New Feature**: Ability to set a project Alias for virtual endpoint URL persistence
- **New Feature**: Export of all of the project’s configurations in a single file
- **Experimental**: Ability to process data in non CSV formats using Ontotext Refine CLI

7.3.1 Improvements and Bugfixes

- OR-397 Cleaned Docker image vulnerabilities
- Replaced documentation section on RDF-izing Tabular Data

7.3.2 Known Issues

Querying extremely large tables via the SPARQL Editor may cause the browser to freeze/crash. We suggest working on a sample and using the CLI for the full file.

7.4 Refine 1.1

Released: 30 September 2022

Important: Refine 1.1:
- **New Feature**: Refine CLI for automating data pipeline steps.

7.4.1 Improvements and Bugfixes

- Updated OpenRefine to version 3.6.1
- Added support for custom OpenReine extensions (compatible with openrefine 3.6.1)
- Fixed OR-333 Exporting to ODF doesn’t work
- Fixed OR-404 Native app binds itself to localhost and not to hostname
- Fixed OR-423 Mapping UI does not find UI translations (en.js)
7.4.2 Known Issues

• OR-359 - Importing very large datasets using SPARQL federation sometimes fails. This will be fixed with GraphDB 10.0.3

7.5 Refine 1.0

Released: 20 July 2022

Important: Refine 1.0:

• Initial release as a standalone application - Refine is no longer part of Ontotext GraphDB

7.5.1 Improvements

• Initial release as a standalone application

OpenRefine is now a separate application.

• Installers are available for all major OSs
• A Docker image is available
• Remote access to GraphDB is configurable
• Access can be secured using basic auth or GraphDB auth

7.5.2 Known Issues

• OR-79 - ontorefine-cli is not available in Refine 1.0. It will be available in 1.1
• OR-80 - Very large projects (> 200K lines) may not be fully RDF-ized
• OR-133 - Large mapping queries, when serialized can exceed the limits of a request and be rejected
ADDITIONAL RESOURCES

This page has additional resources (links) that can help you work with Refine more efficiently.

- The Ontotext Refine landing page has a number of blogs, webinars, and product news.

Below we list OpenRefine resources. The counts are up to date as of July 2022.

- Official OpenRefine site: https://openrefine.org
- Official Documentation: https://docs.openrefine.org/ including:
  - User Manual
  - GREL Functions
  - Technical Reference
- Extra documentation including:
  - Documentation for users
  - Documentation for developers
  - Support options
  - Online courses
  - The book: “Using OpenRefine”
- Wiki with 123 pages, in particular:
  - Recipes where folks might have solved a similar problem as yours
  - External Tutorials in English (124), Spanish (12), German (18), French (4), Japanese (8)
  - Wiki Documentation For Developers

Reconciliation:

- Reconciling in the user manual
- Reconcilable Data Sources in the wiki
- Reconciliation Testbench with a longer list of Recon services
- Reconciliation Service for Knowledge Graph Enrichment with OntoRefine webinar by Ontotext using GraphDB 9.10 and OntoRefine.

Extra tutorials:

- Google Refine Tutorial. David Huynh. NICAR 2011. A must read tutorial by the original creator of Refine.
- Data Carpentry Open Refine Demo. Erika Mudrak, Jeramia Ory, Emily Davenport. Data Carpentry Workshop, Cornell Statistical Consulting Unit. 2015-01
Search for videos on YouTube:
  • OpenRefine videos
  • OpenRefine and RDF videos