Reports WebData - SPARQL and Graph Databases

Query and update graph databases such as Ontotext GraphDB, Apache Jena Fuseki and Stardog from a solution with SPARQL, and get the results as a DataTable.

Reference Modules Reports UI WebData SPARQL and Graph Databases

New in 10.1.6a. SPARQL methods run on a REST WebData row. There is no new module and no new API mode: one row holds the endpoint, the authorization and the query.

A graph database stores data as triples (subject, relation, object), for example `Reactor1 feedsInto Filter1`. SPARQL is the W3C query language for those triples, and every SPARQL 1.1 server answers it at an endpoint URL. FrameworX acts as a SPARQL 1.1 client: a script sends the query to the endpoint and receives the result as a DataTable, ready for a DataTable tag and a DataGrid on a Display.

A question that follows several relationships, such as all equipment downstream of the unit in alarm, becomes one query. To send the UNS itself to a graph database, see Export your UNS to RDF/OWL/GraphDB. For a step-by-step solution, see GraphDB SPARQL Query Example.

Configure the WebData Row
Script Methods
Result Table
Errors and Status
Notes

Configure the WebData Row

In **Reports WebData**, create one row per endpoint. A server that has separate URLs for queries and updates needs two rows, one for each.

Column	Value
API	`REST`
Encoding	`Text`
DefaultURL	The SPARQL endpoint of the repository or dataset. See the table below.
Authorization	`NoAuth`, `BasicAuth` or `BearerToken`, as the server requires.
Editor	The SPARQL query or update text. `{Tag.Name}` references are filled in with the tag values when the call runs.

Endpoint URLs

Server	Query endpoint	Update endpoint
Ontotext GraphDB	`http://<host>:7200/repositories/<repositoryId>`	`http://<host>:7200/repositories/<repositoryId>/statements`
Apache Jena Fuseki	`http://<host>:3030/<dataset>/sparql`	`http://<host>:3030/<dataset>/update`
Other SPARQL 1.1 servers	See the server's documentation for its query and update URLs.	

Script Methods

Call the methods from a Script Task or a Script Class with `await`.

Method	Returns	Use
`SparqlQueryTableAsync()` `SparqlQueryTableAsync(query)`	`DataTable`	Run a SELECT, ASK, CONSTRUCT or DESCRIBE query and get the result as a table.
`SparqlQueryAsync()` `SparqlQueryAsync(query)`	`string`	Run a query and get the server's raw response text, for example SPARQL results JSON.
`SparqlUpdateAsync()` `SparqlUpdateAsync(update)`		Send a SPARQL UPDATE, such as `INSERT DATA` or `DELETE DATA`, to write to the graph.

Editor text or script text

- **Without an argument**, the method sends the Editor text of the row, with every `{Tag.Name}` reference filled in.
- **With an argument**, the method sends that text exactly as given. `{Tag.Name}` references in it are not filled in, so build the value into the string in C#.

Braces in the query body, such as `WHERE { }` or `{}`, are sent unchanged.

```
try
{
    // Editor query, with {Tag.Name} references filled in
    DataTable t = await @Report.WebData.PlantGraph.SparqlQueryTableAsync();
    @Tag.Downstream = t;

    // Query text from the script, sent as written
    string unit = "Reactor1";
    DataTable t2 = await @Report.WebData.PlantGraph.SparqlQueryTableAsync(
        "PREFIX ex: <http://example.com/plant#> " +
        "SELECT ?e WHERE { ex:" + unit + " ex:feedsInto ?e }");

    // Write to the graph through the update endpoint row
    await @Report.WebData.PlantGraphUpdate.SparqlUpdateAsync(
        "PREFIX ex: <http://example.com/plant#> " +
        "INSERT DATA { ex:Packer1 ex:feedsInto ex:Warehouse1 . }");
}
catch (Exception ex)
{
    @Tag.Err = ex.GetType().Name + ": " + ex.Message;
}
```

Result Table

Query form	DataTable shape
SELECT	One column per variable, one row per solution. A variable with no value in a row, for example from `OPTIONAL`, is null.
ASK	One column, `Result`, and one row with `True` or `False`.
CONSTRUCT, **DESCRIBE**	One row per triple, with the columns `Subject`, `Predicate`, `Object`, `ObjectType` (`uri` or `literal`), `Datatype` and `Language`. The response is read from Turtle, N-Triples, JSON-LD or RDF/JSON.

Literal values keep the text the server sent. A date and time such as `2026-10-22T10:15:30.250+02:00` arrives character for character, with its fraction and offset.

Errors and Status

Situation	Result
The server answers with an HTTP error, for example a query with a syntax error	`HttpRequestException`, message starting with the HTTP code (for example `HTTP 400`). `LastStatus` = 18.
The endpoint is unreachable	`HttpRequestException`. `LastStatus` = 18.
The response cannot be read, for example malformed JSON	`FormatException`. `LastStatus` = 12.
The response is RDF/XML	`NotSupportedException`. `LastStatus` = 12.
A JSON-LD response with a remote (URL) `@context`	Fails with a message naming the remote context.
The row's API is not REST	`InvalidOperationException`, for example `SparqlQueryTableAsync is only valid when API=REST. This object's API is GraphQL.`
Empty update text	`InvalidOperationException`: `The SPARQL update is empty. Pass the text to SparqlUpdateAsync.` Nothing is sent.
The row is disabled (`Disable` = 1)	The method returns null and sends nothing.

A call that fails before sending (wrong API, empty URL or empty text) keeps the previous `LastStatus`.

Notes

- The whole result loads into memory as one DataTable. Use `LIMIT` on queries that can return many rows.
- RDF/XML responses are not supported. Ask the server for Turtle, N-Triples, JSON-LD or RDF/JSON.
- Verified against Apache Jena Fuseki 4.10 (queries, updates and every result format, on .NET Framework 4.8 and .NET 10 runtimes) and Ontotext GraphDB 10.8 (SELECT queries).

In this section...
