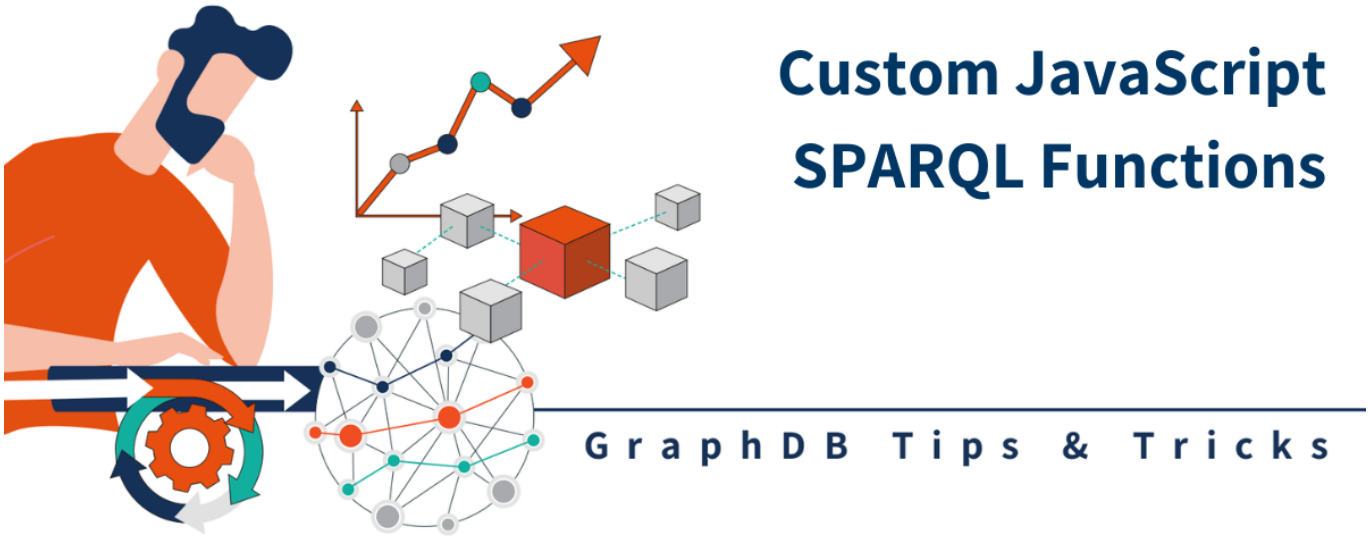
ontotext




Ontotext GraphDB's support for JavaScript means that you can write functions in this popular language, register them with a GraphDB repository, and then call them from SPARQL queries that you use with that repository. The complete documentation of this feature is available on the JavaScript functions page of the GraphDB documentation, and I wanted to demonstrate some fun that I had trying this out.

For my experiment, I used the sample Star Wars dataset that you can download from the first paragraph of GraphDB's Visualize and explore documentation page. Here is some sample data about one of the fictional planets in the Star Wars universe:

```
<https://swapi.co/resource/planet/13> a voc:Planet ;
   rdfs:label "Mustafar"^^xsd:string ;
   voc:climate "hot"^^xsd:string ;
   voc:diameter 4200 ;
   voc:film <https://swapi.co/resource/film/6> ;
   voc:orbitalPeriod 412 ;
   voc:population 20000 ;
```

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

```
   voc:terrain "volcanoes, lava rivers, mountains, caves"^^xsd:string
.
```

Let's say I want to hand off the comma-separated `voc:terrain` values to another application, but that application expects the values in that string to be in sorted order. And, because of the space in values like "lava rivers", the receiving application needs each value enclosed in quotes. I also want to pass a descriptive term of "small", "medium", or "large" about the planet's diameter to that receiving application, so I wrote a function to do that mapping as well.

## Writing and registering the functions

First, I developed `sortCSVList()` and `sizeAsWord()` JavaScript functions in a text file that I tested using the Node.js environment's node command line utility. Once I had both functions doing what I wanted, I pasted the following update request into the GraphDB Workbench's query window to register the functions with the Star Wars repository that I was using:

```
prefix extfn:<https://www.ontotext.com/js#>

INSERT DATA {
    [] <https://www.ontotext.com/js#register> '''

       function sortCSVList(listString) {
           // Lose any space around commas
           listString = listString.replace(/\\s*\\,\\s*/g,",");
           termArray = listString.split(',');
           termArray.sort();
           sortedString = '';
           for (var i = 0; i < termArray.length; i++) {
               sortedString += '"' + termArray[i] + '"';
               if (i+1 < termArray.length) {
                   sortedString += ",";
               }
           }
```

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

```
            return sortedString;
        }

        function sizeAsWord(diameter) {
            if (diameter < 10000) {
                result = "small"
            }
            else if (diameter < 13000) {
                result = "medium"
            }
            else {
                result = "large"
            }
            return result

        }
    '''
}
```

(You can read more about registering, managing, and using JavaScript functions in the GraphDB JavaScript functions page documentation page mentioned above.) As this update request on that page shows, registering functions for the active repository is done by inserting a triple with a blank node as its subject, a predicate of a special URI for registering, and then a string value containing the functions as the inserted triple's object.

When I pasted the bodies of the functions that had worked with Node.js into that `INSERT` statement, at first GraphDB wouldn't register them. I eventually found out that because these functions are inside of the big string passed as the inserted triple's object, the backslashes that I had used in the *`sortCSVList`* regular expression needed to be escaped. That's why you see two at a time in the function's regular expression. Once I added those additional backslashes, the functions registered without a problem.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

ontotext

## Calling your new functions from a SPARQL query

Once I registered those functions I could use them in a SPARQL query. My new functions were registered in the https://www.ontotext.com/js# namespace, so I used a *jsfn* prefix in the query when calling each one. This query retrieves the stored `voc:terrain` and `voc:diameter` values for each planet, passes each to one of my new functions, and displays the input and output values.

```
PREFIX voc: <https://swapi.co/vocabulary/>
PREFIX jsfn:<https://www.ontotext.com/js#>

SELECT ?diameter ?diameterDescription ?terrain ?sortedList
WHERE {
   ?planet a voc:Planet ;
           voc:terrain ?terrain ;
           voc:diameter ?diameter . +
    BIND (jsfn:sizeAsWord(?diameter) AS ?diameterDescription)
    BIND (jsfn:sortCSVList(?terrain) AS ?sortedList)
}
```

The following shows the first few of the 42 result rows that this created:

| | diameter | diameterDescription | terrain | sortedList |
|---|---|---|---|---|
| 1 | "9830"^^xsd:integer | "small" | "oceans, savannas, mountains, grasslands" | "grasslands","mountains","oceans","savannas" |
| 2 | "4200"^^xsd:integer | "small" | "volcanoes, lava rivers, mountains, caves" | "caves","lava rivers","mountains","volcanoes" |
| 3 | "0"^^xsd:integer | "small" | "airless asteroid" | "airless asteroid" |
| 4 | "10088"^^xsd:integer | "medium" | "glaciers, mountains, ice canyons" | "glaciers","ice canyons","mountains" |
| 5 | "9100"^^xsd:integer | "small" | "fungus forests" | "fungus forests" |
| 6 | "14920"^^xsd:integer | "large" | "caves, desert, mountains, volcanoes" | "caves","desert","mountains","volcanoes" |

You can see that the diameter values got mapped to description terms appropriately. Also,

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

the new version of the terrain terms lists them alphabetically, with each enclosed in quotes, even when a term such as “lava rivers” or “airless asteroid” includes a space.

When used in a `CONSTRUCT` query or an `INSERT` update request, the terms returned by your JavaScript functions can be used to enhance data quality. They can add data that is useful to people, like the diameter descriptions above, as well as data that is more useful to processes that consume the data, like the quoted lists.

To be honest, I could have mapped the diameter values to those three terms with standard SPARQL that didn’t need any JavaScript help by using nested `IF()` expressions:

```
BIND (IF(?diameter < 10000,"small",IF(?diameter <
13000,"medium","large"))
      AS ?diameterDescription)
```

This is not quite as readable as the JavaScript version, and would get less and less readable as more logic and function calls were added. The important thing is that, using the template of the JavaScript function that I created for this, you can create much more complex JavaScript logic to enhance your RDF data. These can use all the JavaScript functions, data structures, loops, and conditional statements that you like.

## Managing your JavaScript functions

In addition to registering and using JavaScript functions, the documentation page mentioned above shows you how to list all registered functions and delete them (all by using SPARQL queries and update requests, of course) so that you can manage your collection as it grows larger.

This ability to write your own JavaScript functions and call them from your GraphDB queries and update requests can add a lot to your knowledge graph applications. Try it out and let us know how it enhances yours!

**Take a moment to share your experience with GraphDB on Gartner Peer Insights!**

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

# Custom JavaScript SPARQL Functions

Author: Bob DuCharme

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com