THE GEO PROJECT
GEOSPATIAL EDUCATION & OUTREACH

www.geoproject.msstate.edu

GEO TUTORIAL

CALLING ALGORITHMS FROM FIELD CALCULATOR IN QGIS

Krzysztof Raczynski
Kate Grala
John Cartwright

Geosystems Research Institute
Mississippi State University

OCTOBER 2024

This work was supported through funding by the National Oceanic and Atmospheric Administration Regional Geospatial Modeling Grant, Award # NA19NOS4730207.

Geosystems Research Institute, Mississippi State University, Box 9627, Mississippi State, MS 39762-9652

The Geospatial Education and Outreach Project (GEO Project) is a collaborative effort among the Geosystems Research Institute (GRI), the Northern Gulf Institute (a NOAA Cooperative Institute), and the Mississippi State University Extension Service. The purpose of the project is to serve as the primary source for geospatial education and technical information for Mississippi.

The GEO Project provides training and technical assistance in the use, application, and implementation of geographic information systems (GIS), remote sensing, and global positioning systems for the geospatial community of Mississippi. The purpose of the GEO Tutorial series is to support educational project activities and enhance geospatial workshops offered by the GEO Project. Each tutorial provides practical solutions and instructions to solve a particular GIS challenge.

CALLING ALGORITHMS FROM FIELD CALCULATOR IN QGIS

Krzysztof Raczynski (chrisr@gri.msstate.edu)
Kate Grala (kgrala@gri.msstate.edu)
John Cartwright (johnc@gri.msstate.edu)

Geosystems Research Institute,
Mississippi State University

REQUIRED RESOURCES

- QGIS 3+

FEATURED DATA SOURCES

- Dataset used in this tutorial can be downloaded from: https://arcg.is/19Hv851

OVERVIEW

Knowledge about different processing algorithms, how to run them, and when to use them is crucial in GIS analysis. However, when applied regularly this can lead to performance issues due to the large amount of data created. Dataspace can deplete quickly, even when generating temporary layers, and especially when working with complex projects. This is because each time a processing algorithm is used in QGIS it saves the output to a new layer. This leads to many unnecessary layers and chaos in the project and on the hard drive. Due to this fact, it is useful to know how to run algorithms from the field calculator level to quickly compute what is needed and capture the result without the need to create new layers and perform multiple processing and joining tasks. In this tutorial, you will learn a new trick that will improve your processing skills.

Imagine you are an analyst working for a local GIS company in Mississippi. MDOT, which manages highways in the state, is preparing a cost assessment for grass-cutting along all highways. You were given the task of calculating the total area for grass mowing along each highway in Mississippi, depending on their class. You were given the following specifications for grass mowing:

Highway class:	1	2	3	4	5
Mowing buffer (ft):	100	90	60	40	60

This tutorial will show you how to run this analysis from the field calculator level to decrease the amount of work required when performed manually.

STEP 1. ANALYZING THE PROBLEM

To begin with, download the highway data available in the Featured Data Sources above and add it to the new project in QGIS.

We could approach this problem as a simple processing-analysis problem. If we did that, we would have to extract each class of highway from the main shapefile. Then generate a buffer for each case depending on the given specifications. In the next step, we would have to calculate the area for each case and aggregate it to receive the final results. This approach, however, contains multiple steps and repetitive actions. Additionally, we create 11 additional layers: 5 for extracted highway classes, 5 for buffers generated for each class, and a final aggregated layer. When done like this, we would receive the following results:

Highway class:	1	2	3	4	5	Total
Mowing area (km^2):	110.34	302.92	286.27	141.76	18.32	859.60

While this approach works, the entire process is redundant when considering the abilities of the Field Calculator.

STEP 2. CREATING BUFFER AREAS DIRECTLY

Instead of running multiple algorithms, let's perform all the needed actions from the Field Calculator:

A. Open the **Attribute Table** for the highways layer.
B. Open Field Calculator.
C. Prepare a **new field** with the name *buffer_a*, where we will compute the area of buffers for each highway. Set **field type** to *Decimal number (real)*, with **length** *10* and **precision** *3*.
D. In the Expression Creator, we will write a formula to compute buffer areas. This will be a long formula, so we will break it into steps for learning purposes:

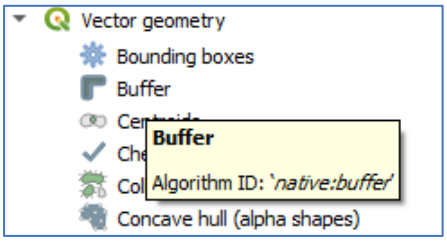

Fig. 1. *By hovering over algorithm in* ***Processing toolbox*** *you can check its name to be called in calculator and outside scripts.*

a. We can call the algorithm by its **ID name** directly in the field calculator window. If you don't know the ID name, hover the mouse over the algorithm in the Processing Toolbox. A small window will be displayed indicating the name (Fig. 1): *buffer*. When calling an algorithm as a function, we need to pass its arguments (parameters) in parenthesis:

```
buffer()
```

b. The suggestion box below the cursor informs us about the order of input parameters: *geometry*, *distance*, *segments*, *cap*, *join*, and *miter_limit*. In Fig. 2, you can see the order is the same as for the regular algorithm when called from the processing toolbox.

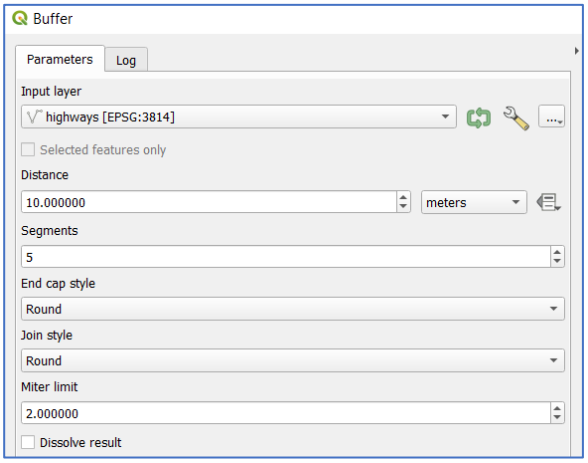

Fig. 2. *The Order of arguments in the algorithm called in* ***Field Calculator*** *resembles the order in the algorithm*

c. We will now fill in the **buffer arguments**, starting with *geometry*. When the algorithm window is called, we can select

a layer to be used. In our case, however, we want to run the tool on self - we are running a field calculator from the highways layer, which we will use to compute the buffer. In such a case we can use *$geometry* to refer to the object's own geometry:

```
buffer($geometry)
```

d. Next, we must set the remaining arguments. As seen in Fig. 2, by default, our layer uses meters as the *distance units*. This means that when calling the function, we need to pass the *distance* argument in meters. Let's use 100 feet for now (30.48 meters):

```
buffer($geometry, 30.48)
```

e. The remaining arguments can be left as in the default settings seen in Fig. 2:

```
buffer($geometry, 30.48, 5, 'Round', 'Round', 2)
```

f. We can see in the Preview below the expression: *<geometry: Polygon>*, which indicates that at this point, our formula returns geometry. We are, however, interested in the area value for the buffer. To compute area, we can wrap our function with another function: *area*. When we do so, we indicate to the program that we want to compute the area of the geometry that we generated with the buffer function:

```
area(buffer($geometry, 30.48, 5, 'Round', 'Round', 2))
```

g. Divide the output by a million to translate the output unit from m^2 to km^2:

```
area(buffer($geometry, 30.48, 5, 'Round', 'Round', 2))/1000000
```

h. There is one issue with the formula we have written so far. If we apply it now, each object will have a 100-foot buffer computed. We have a few conditions to apply depending on the class of the highway. To solve this, we will apply cases. The general construction is as follows:

```
CASE
WHEN condition1 THEN result1
WHEN condition2 THEN result2
...
END
```

Cases can be used to create multiple conditions and outputs within the same expression. We can leverage this construction to modify the buffer area based on the highway class.

i. Create five cases that will calculate buffers depending on the highway class. When you call attribute name, use double quotation marks **"**. You can compare field values directly by calling the attribute name, then the relation sign (e.g., **=** for equality or **>** for greater than), and the value

to compare. This will be your condition, while the result will be the buffer function we created earlier; remember to provide the buffer distance in meters:

```
CASE
WHEN "CLASS" = 1 THEN area(buffer($geometry, 30.480, 5, 'Round', 'Round', 2))/1000000
WHEN "CLASS" = 2 THEN area(buffer($geometry, 27.432, 5, 'Round', 'Round', 2))/1000000
WHEN "CLASS" = 3 THEN area(buffer($geometry, 18.288, 5, 'Round', 'Round', 2))/1000000
WHEN "CLASS" = 4 THEN area(buffer($geometry, 12.192, 5, 'Round', 'Round', 2))/1000000
WHEN "CLASS" = 5 THEN area(buffer($geometry, 18.288, 5, 'Round', 'Round', 2))/1000000
END
```

j. Once the formula is ready and the preview shows a number instead of an error, click **OK**.
k. With the above formula, we have computed the area of each highway buffer, depending on its class.

STEP 3. CREATING AGGREGATED RESULTS

We have calculated buffer areas for each object; now we will **aggregate** the results to receive numbers based on the *CLASS* field:

A. Open **Field Calculator** again and create a **new field** called *total_a*, of *decimal* **type** (10,3).
B. We will now filter all results for each class based on the *CLASS* field:
   a. To create the final value, we first need to filter only the values that match the *CLASS* of an object. To do so, use the *array_agg* function, which creates an array (list) of all the values that match the *CLASS* of the object for which that function is called. This means that we will have repetitions of the same values for each case, where the class of highway is the same:

```
array_agg()
```

   b. The function accepts one required argument: **expression or field to aggregate**. In our case, we will aggregate the calculated *buffer_a* field:

```
array_agg("buffer_a")
```

   c. Additionally, we want to apply a filter that will use the *CLASS* attribute to find only cases of the same highway class. We can pass the attribute name to the *group_by* attribute:

```
array_agg("buffer_a", group_by:="CLASS")
```

C. The above function will return a list of values that fit our filter. If you look at **Preview**, you can see results like the following (example for feature *I 55*; the numbers may vary minimally):

```
[45.216, 1.798, 23.33, 24.665, 12.553, 2.183, 0.571]
```

D. This is not exactly the result we are aiming for, as we would like the final number to represent the **sum** of all the elements in the list instead of the list itself. We can **sum** all array elements by calling the *array_sum* function on the array that we want to use:

```
array_sum(array_agg("buffer_a", group_by:="CLASS"))
```

E. With the above function **run**, we now have all the results we need: individual mowing area for each highway in the ***buffer_a*** field, and total mowing area by highway class in ***total_a*** field (Fig. 3). All of this accomplished without creating any additional layers or repetitive manual clicking.

	OBJECTID	HWYNAME	CLASS	buffer_a	total_a
366	8	MS 2	3	3.001	286.207
367	7348	MS 198	4	1.021	141.743
368	2159	MS 19	3	6.933	286.207
369	3485	MS 184	4	1.448	141.743
370	6480	MS 182	4	2.162	141.743
371	122	MS 18	3	11.162	286.207

Fig. 3. *Portion of the resulting table, where* ***buffer_a*** *displays the individual buffer area and* ***total_a*** *displays the total buffer area by class. Note that* ***buffer_a*** *presents an individual mowing area for each object separately, regardless of its CLASS (marked green), while* ***total_a*** *presents the same area for each case of the same CLASS (marked red). This is due to the way the table is organized; while only one number is necessary for each CLASS, the same value is repeated, as results are not aggregated by CLASS since we also need a mowing area for each object. In the above example, value 141.743 represents the total area for the entire CLASS = 4, and no further addition of these values is required (compare the results to the table in Step 1).*

STEP 4 [OPTIONAL]. CREATING RESULTS IN ONE CALL

If we would like to compute all the above steps directly in one column instead of two, we can combine both functions and replace the ***buffer_a*** attribute used in the aggregation step with the ***buffer()*** functions used before:

```
CASE
WHEN "CLASS" = 1
      THEN array_sum(array_agg(area(buffer($geometry, 30.480, 5, 'Round', 'Round', 2)),
      "CLASS"))/1000000
WHEN "CLASS" = 2
      THEN array_sum(array_agg(area(buffer($geometry, 27.432, 5, 'Round', 'Round', 2)),
      "CLASS"))//1000000
WHEN "CLASS" = 3
      THEN array_sum(array_agg(area(buffer($geometry, 18.288, 5, 'Round', 'Round', 2)),
      "CLASS"))//1000000
WHEN "CLASS" = 4
      THEN array_sum(array_agg(area(buffer($geometry, 12.192, 5, 'Round', 'Round', 2)),
      "CLASS"))//1000000
WHEN "CLASS" = 5
      THEN array_sum(array_agg(area(buffer($geometry, 18.288, 5, 'Round', 'Round', 2)),
      "CLASS"))//1000000
END
```

We have reached the end of this GEO Tutorial. You can use the concepts presented here to explore more functions available in the field calculator. Practice using these concepts on projects you are currently working and remember that the same functionality can be successfully applied to other geometry types. Use this tutorial as your guide and have fun leveraging the power of the field calculator on your own!