Triggering Reports

Reports, like some other entities, require to be processed in order to have their data ready for use. The processing of a report is triggered in the same fashion as any other processed entity, which is the subject of the previous section.

Before processing a report, the report entity must be properly configured. This section will cover how to set up a report using the REST interface.

To successfully set up a report entity, it is necessary to:

1. Define which report program will be used
2. Provide values to the program’s inputs
3. Define a 3D region where the report will run
4. The Report Program Entity

A report entity will obtain its data by running a Python program or a .Net assembly over a custom mashup of spatial datasets. It is up to the report code to decide which inputs will be used and how the data will be interpreted.

In order to set up a report entity, it is necessary to have a report program entity. The program entity holds the Python program, and it can be used to create many different report entities, each one containing the results of a different run of the same program, but with different input choices.

If there is no program entity yet, one must be created with the following properties:

code_origin	Contains the type of code used by the program. Currently, two types are supported: • "PYTHON" - The code is a Python program • "NETASM" - The code is provided as a compiled .Net assembly
code	If code_origin is "PYTHON", this property contains the Python code for the program, in Base64 format. If code_origin is "NETASM", this property identifies the entry-point class for the Lambda program in the form: <FullyNamespacedClassName>:<DLLName>

program_type	Contains the type of the program. The program type can be one of the following: • “VOXEL” – The program is for a voxel entity generator. This type only accepts "PYTHON" as code_origin. • “VIEW” – The program creates a view This type only accepts "PYTHON" as code_origin. • “REPORT” – The program creates a report. Reports can be have "PYTHON" or "NETASM" as code_origin

For Python programs, the supplied Python code can be validated by making an additional REST call. See the Validating a Program section.

For .Net assemblies, the entity must be sent for processing so the platform can verify the supplied code. See the Triggering Jobs section for information on how to start the processing of the program entity.

The entity creation request will return the ID of the report program in the system. This ID will be later used to set up the report entity.

Providing input values

In most cases a report program will request input from the user. In the context of a REST call, it may not be possible to prompt the user for any information, so it is up to the REST caller to supply the values the report program expects. This requires knowledge of which inputs the report program expects.

The report program may call any of the Voxel Farm Python interface functions to ask for input. These functions are:

voxelfarm.input(id, ...)	Requests a numeric value
voxelfarm.input_string(id, ...)	Requests a string
voxelfarm.input_date(id, ...)	Requests a date/time
voxelfarm.input_entity(id, ...)	Requests the unique ID of an existing entity in the project
voxelfarm.input_attributes(id, ...)	Requests an attribute set for an existing entity
voxelfarm.input_query(id, ...)	Requests a filter query for an existing entity

When using the .Net assembly interface, the input functions follow the same pattern. The same rules apply.

The first parameter in every call is an alphanumeric identifier. This identifier is used to store the input value in the entity model. If caller of the REST interface is aware of the calls made by the program requesting input, for each call to an input function, a property must be added to the report entity containing the value for the input.

input_value_#id#	Contains the value for the input. Replace the #id# placeholder by the actual id used by the input.
input_label_#id#	Contains the label for the input. Replace the #id# placeholder by the actual id used by the input.
input_type_#id#	Specifies type of the input. Replace the #id# placeholder by the actual id used by the input. The possible input types are: 0 – Numeric 1 – Date 2 – String 3 – Entity 4 – Attribute Set 5 – Query Expression 6 – Boolean (Checkbox) 7 – Color Legend 8 – Drill-hole Settings
input_filter_#id#	For Specifies type of the input. Replace the #id# placeholder by the actual id used by the input. Must be non-zero for type 3 (Entity), contains a numeric value containing a binary mask identifying which types of entities can be potentially selected. Use binary OR to merge different types into a single mask. The values identifying each identity type are (in hexadecimal): 0x01 – Voxel Terrain 0x02 – Voxel Block Model 0x04 – Voxel Generator 0x08 – Realtime Voxel Terrain 0x10 – Indexed Point Cloud

The caller of the REST interface may not be aware of which inputs the report program expects. It is possible to discover the list of inputs by performing an additional REST call. This REST call is covered in the “Getting program inputs” section.

Providing a 3D region

The system currently supports one type of 3D region, which is defined as a horizontal 2D polygon that has been extruded along the vertical dimension.

A region can be expressed in the REST interface as a single string, containing the following format:

1,<min_height>,<max_height>,<polygon_point0_x>, <polygon_point0_y>, <polygon_point1_x>, <polygon_point1_y>, ...

The format is a list of numbers separated by commas. The first number must be 1, this identifies this is an extruded polygon region. The second two numbers are world coordinates for the vertical bounds of the region. The following numbers are the X, Y coordinates of each region point. The polygon points are defined in counter-clockwise order.

For example, the following region string defines a 10-unit cube centered in the project's origin:

1,-5,5,-5, 5,5,5,5,-5,-5,-5

Putting the Report entity together

Once the report program and its inputs are known, it is possible to define all properties required to create the report entity. The following table lists these properties:

name	Set to the readable name of the report.
description	Set to a textual description of the report.
type	Must be set to "FILE".
state	Set to "PARTIAL"
file_type	Set to "REPORT"
file_folder	Set to the ID of the folder that will contain the report. Set to "0" to use the project's root folder.
program	Set to the ID of the program entity that contains the report code
lod	Set to zero.
region	Set to region string. This is the region where the scope will run.
input_value_#id#	For each input in the program, se to the value for the input. Replace the #id# placeholder by the actual id used by the input.

input_label_#id#	For each input in the program, set to the label for the input. Replace the #id# placeholder by the actual id used by the input.
input_type_#id#	For each input in the program, set to type of the input. Replace the #id# placeholder by the actual id used by the input. The possible input types are: 0 – Numeric 1 – Date 2 – String 3 – Entity 4 – Attribute Set 5 – Query Expression
input_filter_#id#	Must defined for each input in the program. Replace the #id# placeholder by the actual id used by the input. Must be non-zero for type 3 (Entity), contains a numeric value containing a binary mask identifying which types of entities can be potentially selected. Use binary OR to merge different types into a single mask. The values identifying each identity type are (in hexadecimal): 0x01 – Voxel Terrain 0x02 – Voxel Block Model 0x04 – Voxel Generator 0x08 – Realtime Voxel Terrain 0x10 – Indexed Point Cloud

The next step is to create the entity using the values defined above. See the “Creating an entity” section for instruction on how to perform this operation.

Right after it is created, the report entity will be in unprocessed state. To trigger the report’s processing, follow the steps outlined in the “Triggering jobs” section.

Report example

This example will use the REST interface to create a report that computes the volume of a large sphere. Both the report and the sphere will be Python programs. This example does not rely on any external data or entities that may already exist in the project.

This is the Python code for the sphere object:

```
import voxelfarm as vf
import math
vf.init()
x = vf.input("x", "Sphere Center X")
```

```
y = vf.input("y", "Sphere Center Y")
z = vf.input("z", "Sphere Center Z")
r = vf.input("r", "Sphere Radius")
vf.set_entity_bounds_x(x - r, x + r)
vf.set_entity_bounds_y(y - r, y + r)
vf.set_entity_bounds_z(z - r, z + r)
for v in vf.field:
    p = vf.get_field_origin(v)
    dx = x - p[0]
    dy = y - p[1]
    dz = z - p[2]
    d = math.sqrt(dx * dx + dy * dy + dz * dz)
    vf.set_field(v, d - r)
```

This program must be converted to Base64 before it can be stored as a Program entity:


```
aW1wb3J0IHZveGVsZmFybSBhcyB2ZgppbXBvcnQgbWF0aAoKdmYuaW5pdCgpCgp4ID0gdmYuaW5wdXQoIngiLCAiU3BoZXJlIENlbnRlciBYIikKeSA9IHZmLmlucHV0KCJ5IiwgIlNwaGVyZSBDZW50ZXIgWSIpCnogPSB2Zi5pbnB1dCgieiIsICJTcGhlcmUgQ2VudGVyIFoiKQpyID0gdmYuaW5wdXQoInIiLCAiU3BoZXJlIFJhZGl1cyIpCgp2Zi5zZXRfZW50aXR5X2JvdW5kc194KHggLSByLCB4ICsgcikKdmYuc2V0X2VudGl0eV9ib3VuZHNfeSh5IC0gciwgeSArIHIpCnZmLnNldF9lbnRpdHlfYm91bmRzX3ooeiAtIHIsIHogKyByKQpmb3IgdiBpbiB2Zi5maWVsZDoKICAgIHAgPSB2Zi5nZXRfZmllbGRfb3JpZ2luKHYpCiAgICBkeCA9IHggLSBwWzBdCiAgICBkeSA9IHkgLSBwWzFdCiAgICBkeiA9IHogLSBwWzJdCiAgICBkID0gbWF0aC5zcXJ0KGR4ICogZHggKyBkeSAqIGR5ICsgZHogKiBkeikKICAgIHZmLnNldF9maWVsZCh2LCBkIC0gcik=
```


The following REST call creates the Program entity for the sphere:

```
Invoke-WebRequest -Uri "http://localhost:58697/entity.ashx?project=myproject" -Method "POST" -Headers @{"Origin"="http://localhost:58697"; "Accept-Encoding"="gzip, deflate, br"; "Accept-Language"="en-US,en;q=0.9,es;q=0.8"; "User-Agent"="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36"; "Accept"="*/*"; "Referer"="http://localhost:58697/cloud/project.html?id=myproject"} -ContentType "application/json; charset=UTF-8" -Body
"{`"name`":`"Sphere`",`"type`":`"FILE`",`"state`":`"COMPLETE`",`"code`":`"aW1wb3J0IHZveGVsZmFybSBhcyB2ZgppbXBvcnQgbWF0aAoKdmYuaW5pdCgpCgp4ID0gdmYuaW5wdXQoIngiLCAiU3BoZXJlIENlbnRlciBYIikKeSA9IHZmLmlucHV0KCJ5IiwgIlNwaGVyZSBDZW50ZXIgWSIpCnogPSB2Zi5pbnB1dCgieiIsICJTcGhlcmUgQ2VudGVyIFoiKQpyID0gdmYuaW5wdXQoInIiLCAiU3BoZXJlIFJhZGl1cyIpCgp2Zi5zZXRfZW50aXR5X2JvdW5kc194KHggLSByLCB4ICsgcikKdmYuc2V0X2VudGl0eV9ib3VuZHNfeSh5IC0gciwgeSArIHIpCnZmLnNldF9lbnRpdHlfYm91bmRzX3ooeiAtIHIsIHogKyByKQpmb3IgdiBpbiB2Zi5maWVsZDoKICAgIHAgPSB2Zi5nZXRfZmllbGRfb3JpZ2luKHYpCiAgICBkeCA9IHggLSBwWzBdCiAgICBkeSA9IHkgLSBwWzFdCiAgICBkeiA9IHogLSBwWzJdCiAgICBkID0gbWF0aC5zcXJ0KGR4ICogZHggKyBkeSAqIGR5ICsgZHogKiBkeikKICAgIHZmLnNldF9maWVsZCh2LCBkIC0gcik=`",`"program_type`":`"VOXEL`",`"file_type`":`"PROGRAM`",`"file_size`":`"0`",`"file_date`":`"1557157137959`",`"file_folder`":`"0`"}"
```

The server returns the ID of the newly created program:

{"result" : "success", "id" : "03318353117A40D6AA8B6C09A85A1060"}

In a similar fashion, it is necessary to create a program entity for the report. The following report program computes the volume of an object:

```
import voxelfarm as vf
entity = vf.input_entity("entity1", "Entity", vf.type.voxel_terrain | vf.type.voxel_generator)
voxels = vf.load_voxels(entity, vf.attribute.volume, None)
volume = 0.0
for v in vf.voxels:
    volume += vf.get_volume(voxels, v)
sum = vf.start_sum("Item", "Object Volume")
vf.sum(sum, volume)
```

This program must be converted to Base64 before it can be stored as a Program entity:

aW1wb3J0IHZveGVsZmFybSBhcyB2ZgplbnRpdHkgPSB2Zi5pbnB1dF9lbnRpdHkoImVudGl0eTEiLCAiRW50aXR5IiwgdmYudHlwZS52b3hlbF90ZXJyYWluIHwgdmYudHlwZS52b3hlbF9wcm9ncmFtKQp2b3hlbHMgPSB2Zi5sb2FkX3ZveGVscyhlbnRpdHksIHZmLmF0dHJpYnV0ZS52b2x1bWUsIE5vbmUpCnZvbHVtZSA9IDAuMApmb3IgdiBpbiB2Zi52b3hlbHM6CiAgICB2b2x1bWUgKz0gdmYuZ2V0X3ZvbHVtZSh2b3hlbHMsIHYpCnN1bSA9IHZmLnN0YXJ0X3N1bSgiSXRlbSIsICJPYmplY3QgVm9sdW1lIikKdmYuc3VtKHN1bSwgdm9sdW1lKQo=

The following REST call creates the program entity for the report:

```
Invoke-WebRequest -Uri "http://localhost:58697/entity.ashx?project=myproject" -Method "POST" -Headers @{"Origin"="http://localhost:58697"; "Accept-Encoding"="gzip, deflate, br"; "Accept-Language"="en-US,en;q=0.9,es;q=0.8"; "User-Agent"="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36"; "Accept"="*/*"; "Referer"="http://localhost:58697/cloud/project.html?id=myproject"} -ContentType "application/json; charset=UTF-8" -Body "{`"name`":`"Compute Volume`",`"type`":`"FILE`",`"state`":`"COMPLETE`",`"code`":`"aW1wb3J0IHZveGVsZmFybSBhcyB2ZgplbnRpdHkgPSB2Zi5pbnB1dF9lbnRpdHkoImVudGl0eTEiLCAiRW50aXR5IiwgdmYudHlwZS52b3hlbF90ZXJyYWluIHwgdmYudHlwZS52b3hlbF9wcm9ncmFtKQp2b3hlbHMgPSB2Zi5sb2FkX3ZveGVscyhlbnRpdHksIHZmLmF0dHJpYnV0ZS52b2x1bWUsIE5vbmUpCnZvbHVtZSA9IDAuMApmb3IgdiBpbiB2Zi52b3hlbHM6CiAgICB2b2x1bWUgKz0gdmYuZ2V0X3ZvbHVtZSh2b3hlbHMsIHYpCnByaW50KCJQYmplY3QgVm9sdW1lIikKdmYuc3VtbWFyeSgpCg==`",`"program_type`":`"REPORT`",`"file_type`":`"PROGRAM`",`"file_size`":`"0`",`"file_date`":`"1557157689680`",`"file_folder`":`"0`"}"
```

The server returns the ID of the new report program:

{"result" : "success", "id" : "33ED105A348F44DBADB146867B383E71"}

The next step is to create an actual sphere object. The earlier step only created a program that can produce any sphere given some input parameters. In this step, a new entity must be created which links to the sphere program and provides values for the inputs the sphere program expects.

By examining the Sphere program, it can be seen it requests four different inputs:

```
x = vf.input("x", "Sphere Center X")
y = vf.input("y", "Sphere Center Y")
z = vf.input("z", "Sphere Center Z")
r = vf.input("r", "Sphere Radius")
```

The inputs are identified by the “x”, “y”, “z” and “r” input identifiers. These identifiers will be used in the next REST call to supply the values expected by the Sphere program.

The following REST request creates a Sphere entity centered at the project’s origin and with a radius of 50 meters:

```
Invoke-WebRequest -Uri "http://localhost:58697/entity.ashx?project=myproject" -Method "POST" -Headers @{"Origin"="http://localhost:58697"; "Accept-Encoding"="gzip, deflate, br"; "Accept-Language"="en-US,en;q=0.9,es;q=0.8"; "User-Agent"="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36"; "Accept"="*/*"; "Referer"="http://localhost:58697/cloud/project.html?id=myproject"} -ContentType "application/json; charset=UTF-8" -Body "{`"name`":`"My Sphere`",`"type`":`"FILE`",`"state`":`"COMPLETE`",`"program`":`"03318353117A40D6AA8B6C09A85A1060`",`"file_type`":`"VOXGEN`",`"file_size`":`"0`",`"file_date`":`"1557158594400`",`"file_folder`":`"0`",`"input_count`":`"4`",`"input_label_x`":`"Sphere Center X`",`"input_filter_x`":`"0`",`"input_type_x`":`"0`",`"input_value_x`":`"0`",`"input_label_y`":`"Sphere Center Y`",`"input_filter_y`":`"0`",`"input_type_y`":`"0`",`"input_value_y`":`"0`",`"input_label_z`":`"Sphere Center Z`",`"input_filter_z`":`"0`",`"input_type_z`":`"0`",`"input_value_z`":`"0`",`"input_label_r`":`"Sphere Radius`",`"input_filter_r`":`"0`",`"input_type_r`":`"0`",`"input_value_r`":`"50`"}"
```

The server returns the ID of the newly created entity:

{"result" : "success", "id" : "060FCABE28CA4E41B46C89B0F65CB57E"}

At this point, all auxiliary entities required to create the report entity are ready. The next steps will be about setting up the actual report entity.

Like the sphere object, creating the report entity will require providing values to the inputs expected by the report program. From inspecting the report code, it can be seen the reports expects one input of type entity:

entity = vf.input_entity("entity1", "Entity", vf.type.voxel_terrain | vf.type.voxel_generator)

This input is identified as “entity1” and it will receive the ID of the sphere entity previously created so the report inspects the data produced by the sphere and gets to compute its volume. The type for this input is set to 3 (Entity) and the filter is set to 5 (Voxel Terrain + Voxel Generator).

The report entity also requires a region string. The following region defines a bounding box that encloses the sphere:

1,-50,50,-50,50,50,50,50,-50,-50,-50

The following REST request creates the report entity:

```
Invoke-WebRequest -Uri "http://localhost:58697/entity.ashx?project=myproject" -Method "POST" -Headers @{"Origin"="http://localhost:58697"; "Accept-Encoding"="gzip, deflate, br"; "Accept-Language"="en-US,en;q=0.9,es;q=0.8"; "User-Agent"="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36"; "Accept"="*/*"; "Referer"="http://localhost:58697/cloud/project.html?id=myproject"} -ContentType "application/json; charset=UTF-8" -Body "{`"name`":`"Volume of Sphere`",`"type`":`"FILE`",`"state`":`"PARTIAL`",`"program`":`"33ED105A348F44DBADB146867B383E71`",`"file_type`":`"REPORT`",`"file_size`":`"0`",`"file_date`":`"1557161627287`",`"file_folder`":`"0`",`"input_count`":`"1`",`"region`":`"1,-50,50,-50,50,50,50,50,-50,-50,-50`",`"lod`":`"0`",`"input_label_entity1`":`"Entity`",`"input_filter_entity1`":`"5`",`"input_type_entity1`":`"3`",`"input_value_entity1`":`"060FCABE28CA4E41B46C89B0F65CB57E`"}"
```

The server returns the ID for the report entity:

{"result" : "success", "id" : "08A1E6851DFD46F3B0EC958D1118A823"}

The last call created a new entity for the report, however the results for the report will not be ready until the report entity has been triggered for processing and the processing is complete.

The following REST request triggers the processing of the report:

```
Invoke-WebRequest -Uri
"http://localhost:58697/events.ashx?project=myproject&id=08A1E6851DFD46F3B0EC958D1118A823&org=2343243456
678890" -Method "POST" -Headers @{"Origin"="http://localhost:58697"; "Accept-Encoding"="gzip, deflate, br"; "Accept-
Language"="en-US,en;q=0.9,es;q=0.8"; "User-Agent"="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36"; "Accept"="*/*";
"Referer"="http://localhost:58697/cloud/project.html?id=myproject"} -ContentType "multipart/form-data; boundary=----
WebKitFormBoundaryZATmyEvaQ9XPOzuY" -Body ([System.Text.Encoding]::UTF8.GetBytes("------
WebKitFormBoundaryZATmyEvaQ9XPOzuY$([char]13)$([char]10)Content-Disposition: form-data;
name=`"process`"$([char]13)$([char]10)$([char]13)$([char]10)RUN_REPORT$([char]13)$([char]10)------
WebKitFormBoundaryZATmyEvaQ9XPOzuY--$([char]13)$([char]10)"))
```

This example does not use the callback URL, it will poll until the state of the report entity is set to “COMPLETE”. The following REST request will retrieve the report entity:

```
Invoke-WebRequest -Uri "http://localhost:58697/entity.ashx?id=08A1E6851DFD46F3B0EC958D1118A823" -Headers
@{"Upgrade-Insecure-Requests"="1"; "User-Agent"="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36";
"Accept"="text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3"; "Accept-Encoding"="gzip, deflate, br"; "Accept-Language"="en-US,en;q=0.9,es;q=0.8"}
```

The server will respond with a JSON object that contains all properties for the report:

```
{"lod":"0","file_folder":"0","input_type_entity1":"3","input_count":"1","input_filter_entity1":"5","file_type":"REPORT","proj
ect":"myproject","name":"Volume of
Sphere","state":"PARTIAL","program":"33ED105A348F44DBADB146867B383E71","file_size":"0","partial_progress":"0","re
gion":"1,-50,50,-50,50,50,50,50,-50,-50,-
50","ID":"08A1E6851DFD46F3B0EC958D1118A823","file_date":"1557162371543","input_value_entity1":"060FCABE28CA
4E41B46C89B0F65CB57E","partial_status":"8,000,000 cubic meter volume
analyzed...","type":"FILE","input_label_entity1":"Entity"}
```

By inspecting the value of the “state” property, it is possible to determine if the entity has completed processing, and whether any errors may have occurred. When the processing completes, the “state” property will switch to either “COMPLETE” or “ERROR”.

Once the report’s processing has completed, the data generated by the report can be downloaded as a CSV file using the REST API:

```
Invoke-WebRequest -Uri
"http://localhost:58697/file.ashx?org=2343243456678890&project=myproject&id=08A1E6851DFD46F3B0EC958D1118A823&filename=report.csv&namehint=Vm9sdW1lb2ZTcGhlcmUuY3N2" -Headers @{"Accept"="*/*"; "Accept-Encoding"="gzip, deflate, br"; "Referer"="http://localhost:58697/cloud/project.html?id=myproject"; "Accept-Language"="en-US,en;q=0.9,es;q=0.8"; "User-Agent"="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36"}
```

The contents of the file are:

Item,Object Volume
Item,523605.137331239

At voxel size 0.5m, the report program has computed the volume of the 50m sphere to be: 523,605.137331239 cubic meters. The analytical result for the volume of a 50m sphere is: 523,598.7755983 cubic meters. The error introduced by the discretization of the sphere is 0.0012%

Revision #2
Created 17 March 2025 15:49:38 by admin
Updated 17 March 2025 21:12:28 by admin