Writing Report Lambdas

Reports are processes that the platform runs over spatial datasets in a massively parallel fashion. The code for these processes can be supplied entirely by users, and can be provided as a Python program or a .Net assembly. This topic covers how to implement a Report process using a .Net assembly and C#.

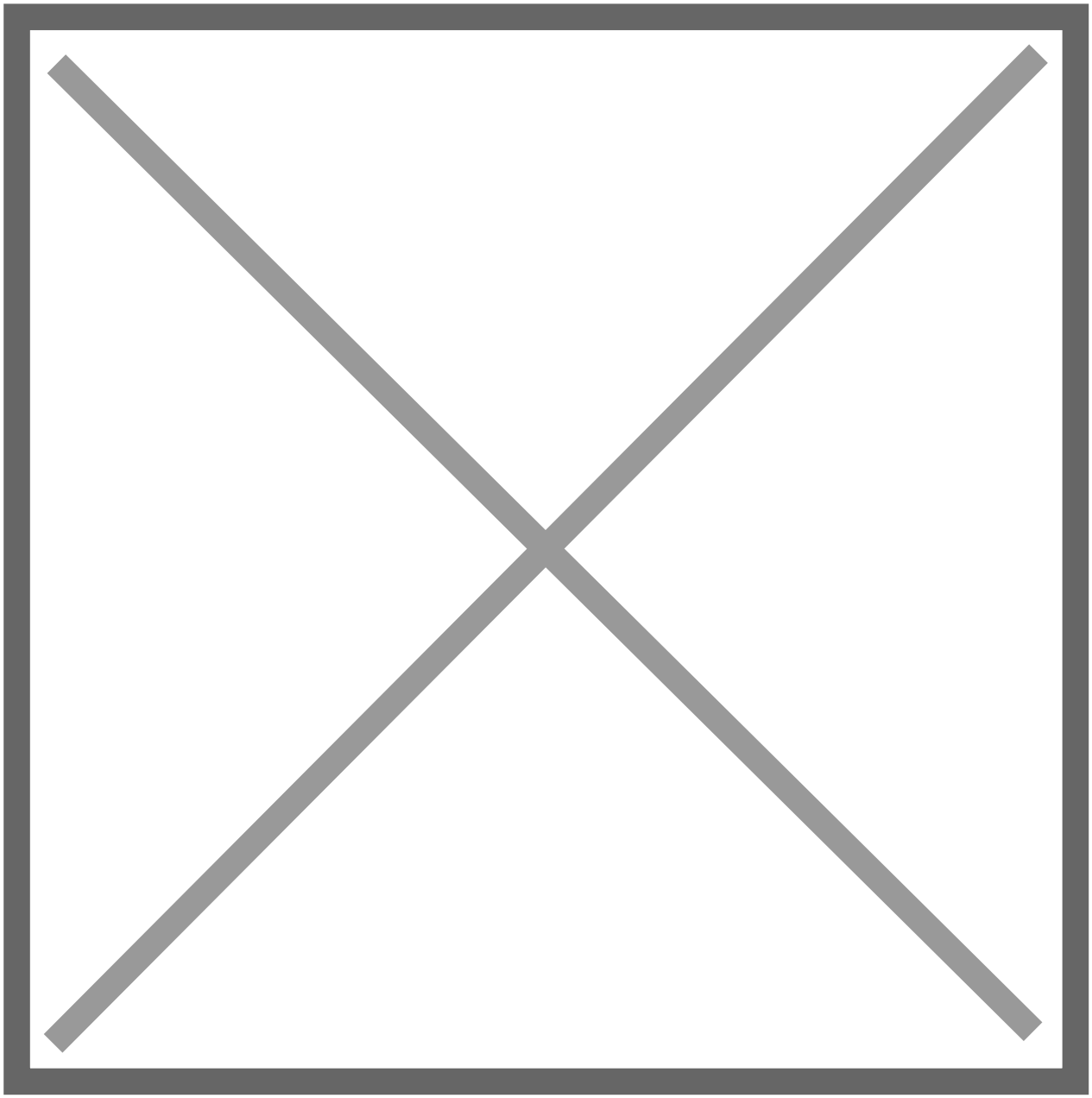
In broad strokes, the user is required to:

1. Create a class in C# that implements the report interface (IVoxelFarmReportLambda)
2. Include that class in a library, and upload the .Net assembly for the library to the spatial project that contains the data to be used in the report
3. Create a report entity that applies the uploaded .Net assembly program to a given region of space

This topic focuses on how a user would create an implementation of the IVoxelFarmReportLambda interface. See the Creating Spatial Lambdas in C# topic to learn more about how to import the lambda code into the platform. The Triggering Reports topic discuses how to start a report over a given spatial region.

Stages of a Spatial Report's execution

Before looking at how an implementation of the report lambda class would be made, it helps to understand the different stages a spatial report will go through:



Stage 1: Gather Inputs. During this stage, the report asks questions to the user or caller. For instance if the report is computing the interaction between two or more datasets, the report code could require these inputs to be provided.

Stage 2: Create Data Mashups. Based on the inputs that were provided in the earlier stage, and following the intent of the report, the report code will combine the datasets in particular ways that are useful to the report. Volumetric datasets can be easily combined by stacking them together, or by performing volumetric boolean operations like Union, Intersection and Complements.

Stage 3: Iterate and analyze data. In this stage, the report iterates over the results of the data mashups and gets to inspect the results. This could lead to the discovery of new facts about the information which the report can accumulate using special devices like Sums and Lists. See the following section for a better understanding of Sums and Lists.

Stage 4: Summarize results. The platform runs earlier stages in a massively parallel fashion. This fact is transparent to the programmer who writes the report, as the report code can be written as if it was running in a fully serial manner. There is, however, one last stage that does not in parallel, and runs once all the parallel processing is complete. During this stage, it is possible to look at the results gathered by the massively parallel stages and produce new facts and data for the report.

Sums and Lists

Report Lambda classes must be stateless. The platform runs the code in a large number of nodes at the same time, for this reason the implementation of the report interface must be purely functional.

Since they are stateless, report lambdas can use two concepts provided by the platform to remember the facts they are discovering: Sums and Lists.

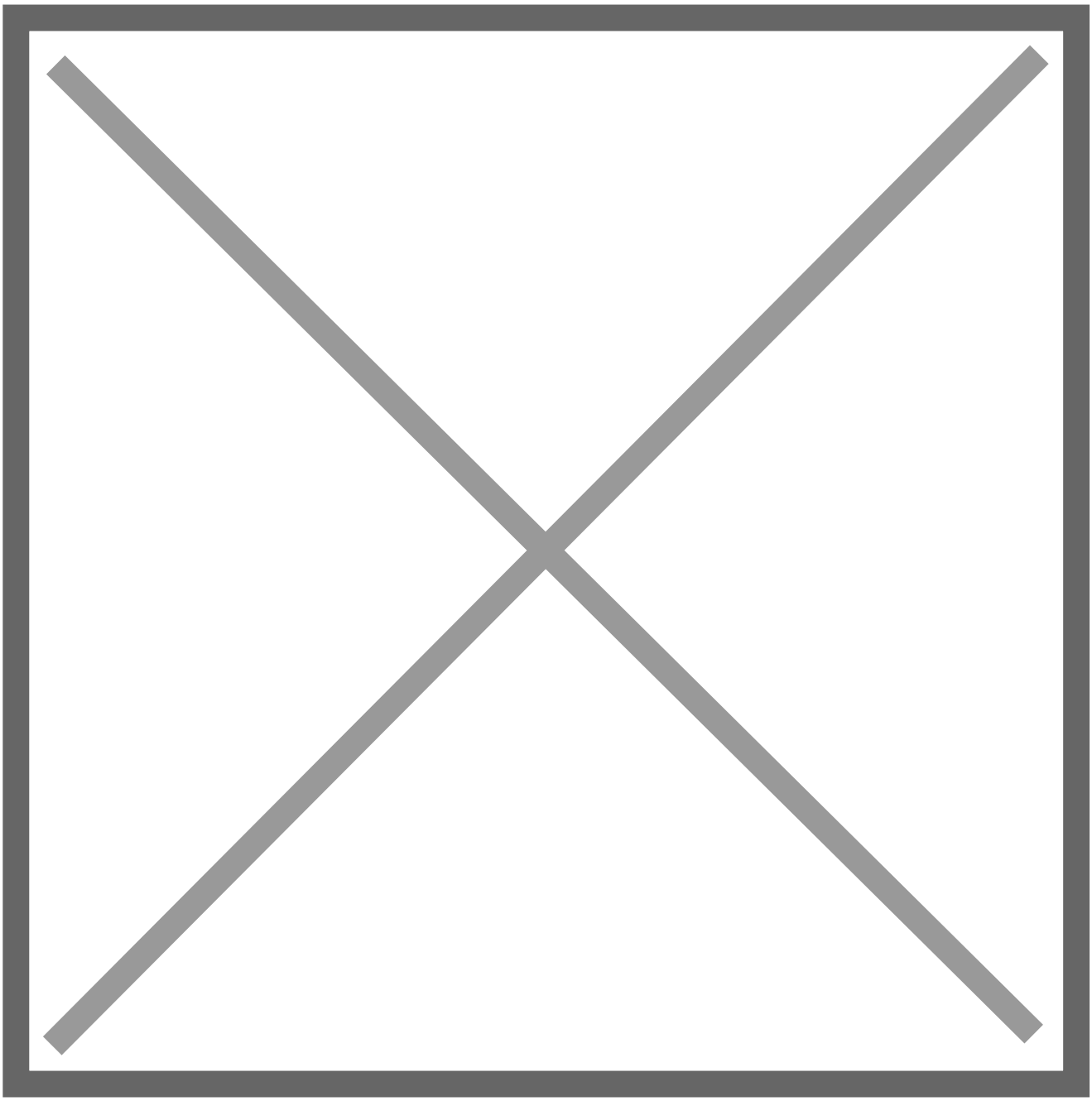
A Sum is a global counter. A report can start as many Sums as it needs. The platform integrates all the partial values for each Sum as they are produced by parallel executions of the spatial lambda that target disjoint regions of space. A simple example of a sum would be how a lambda computes the volume of a very large object. The code would declare a singe sum for the final volume, and then add the volume of each voxel to the sum.

A List allows to collect a series of facts as the lambda executes over space. For instance, assume a report needs to add an entry to an error log wherever certain condition is met by the inspected data. By using a List, the programmer could emit one entry for the list every time an error is detected. Once the report completes, the last stage would be able to access every entry in the list and perform additional actions.

The IVoxelFarmReportLambda interface

A spatial report lambda is required to implement the IVoxelFarmReportLambda interface. This interface is included in the VoxelFarmCloudLibrary assembly.

Implementing the interface requires implementing only two methods, which cover all the execution stages:



1. RunReport: This method takes one parameter of type IVoxelFarmReportLambdaHost, called the lambda's "host". The host provides the functionality required to interact with the platform, including asking for inputs, loading data and creating mashups. In this function, the lambda can gather inputs, create mashups and iterate over the data. This is also where Sums and Lists are created and populated by the lambda code.
2. Done: This method takes one parameter of type IVoxelFarmReportDoneLambdaHost, called "host". The host provides functionality required at this stage, like iterating over list results, sums, and uploading results as files or new entities to the platform.

Revision #2
Created 17 March 2025 19:25:22 by admin
Updated 17 March 2025 19:34:41 by admin