Architecture Document

Introduction

There are four high-level components in the Biomembrane application. These components correlate with the subsystems, below.

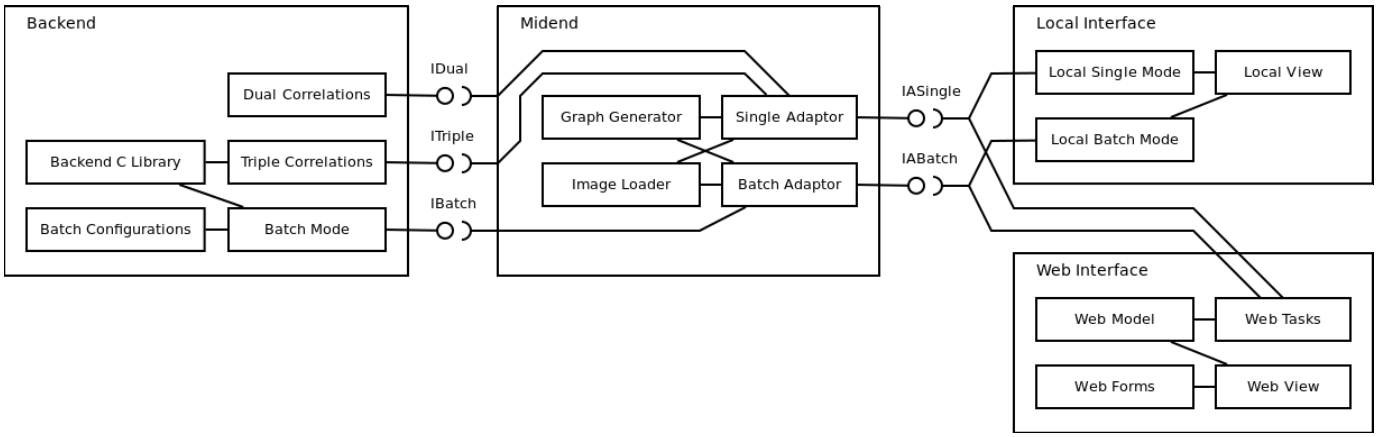
The first component is the backend, which inputs images (as data arrays) with some parameters (integers and floats) and outputs resulting data arrays. The backend is therefore difficult to communicate with as its giving mostly raw data back.

This is where the Midend comes in, acting as a bridge between the backend and the views. The midend takes raw input from the interface and changes it to be ready for the backend. It then wraps the results of the backend calls in a convenient form, which can then be easily used by the frontend to display results or generate graphs.

The Frontend comprises two components, the local interface and web interface. Both use the same basic input (direct user input) and send it to the Midend, retrieving the wrapped result from the midend.

System Design

Software Subsystems



The ICS application consists of four main subsystems: the backend, the midend, the local interface and the web interface. Each of these systems consists of several components. The diagram above shows the structure of the application.

The backend consists of a C library to perform computationally expensive calculations, some sample batch configurations, components to perform dual/triple correlations and lastly a component to perform batch-mode correlations (both dual and triple). It proves interfaces to the correlation components, while the C library and the sample batch configurations used internally.

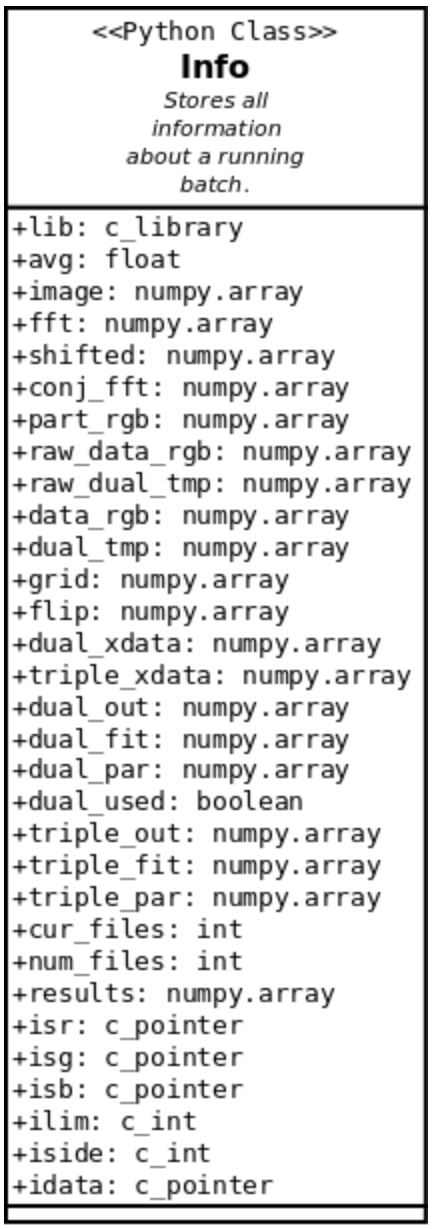
The midend provides a convenient interface for the local and web subsystems to access the backend functionality. It uses the interfaces provided by the backend, and in addition an internal graph generator and image loader, to provide interfaces to its single and batch adaptors, which adapt the backend components with additional functionality and convenience routines.

The local interface provides a user-friendly method to use the application on a local machine. It accepts and validates user input, uses the adaptors provided by the midend to perform calculations and generate output graphs and results, and then displays the graphs and results to the user. It can be run in either single or batch mode. The interfaces for these modes is different, since all input parameters are known beforehand for batch mode, and the batch mode runs on sets of images rather than an individual image.

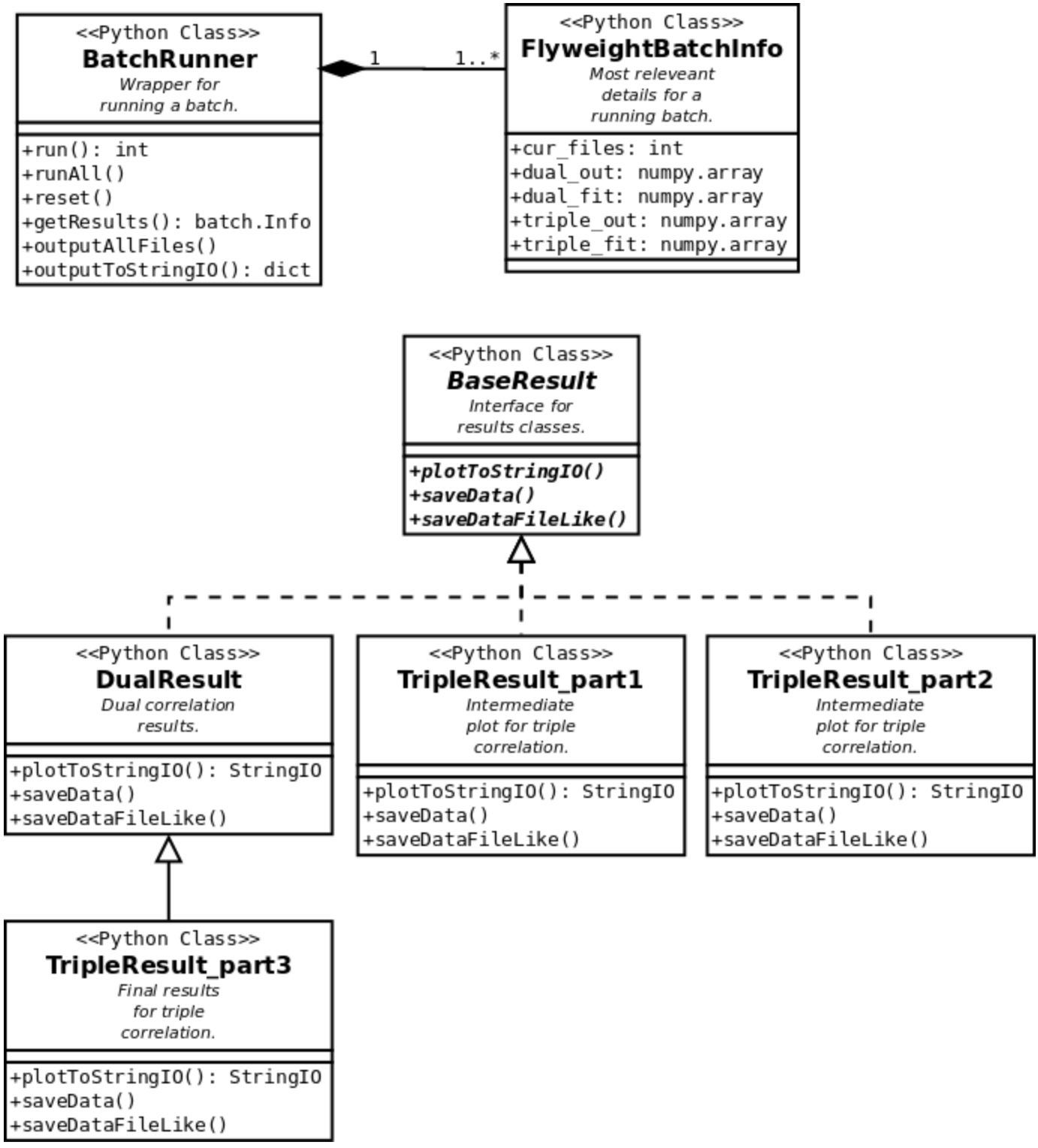
The web interface is similar to the local interface, except that it used across the web, so image files have to be uploaded to the server, and the results have to be sent from the server to the user.

System Domain Model

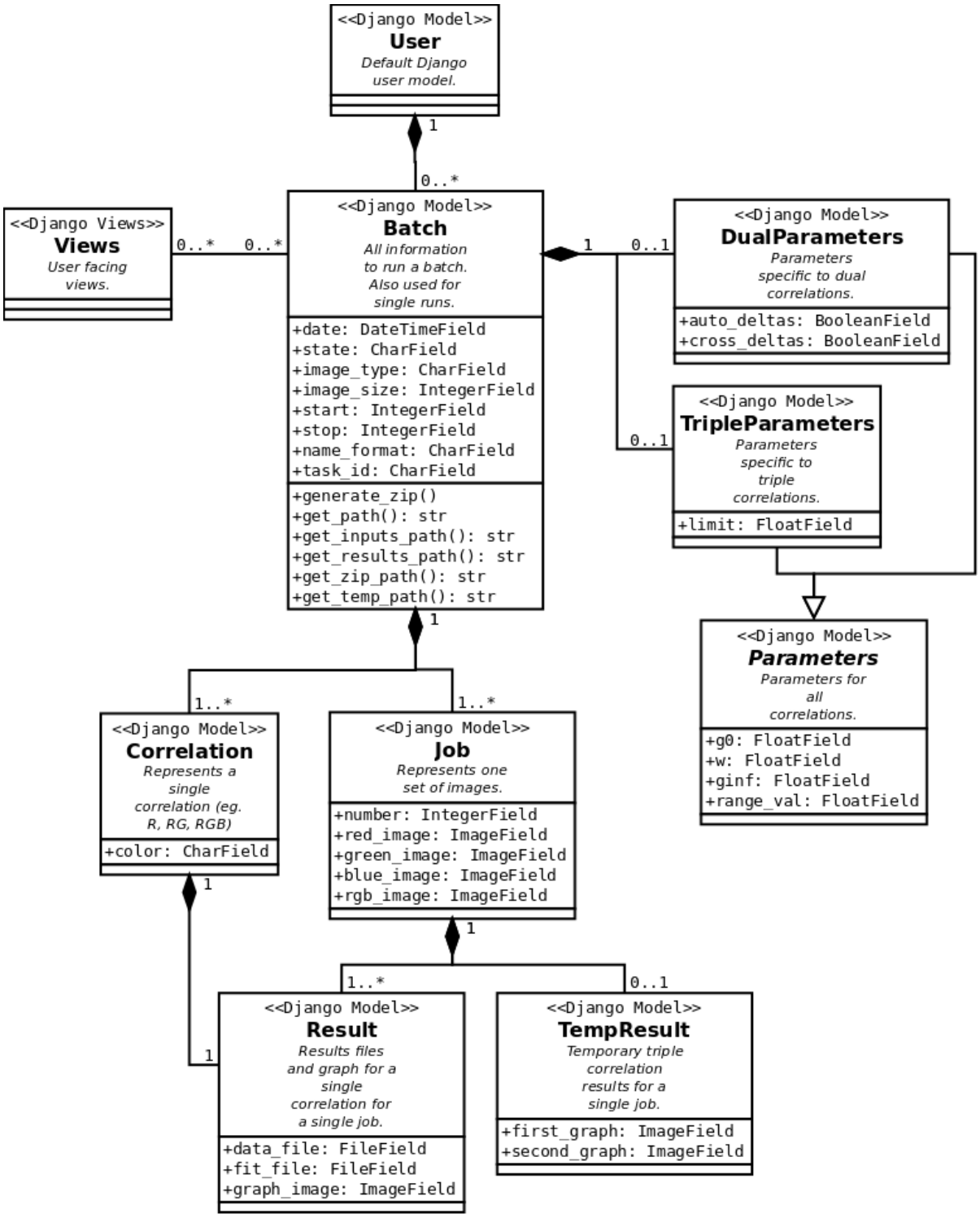
Backend GUI Class Diagram



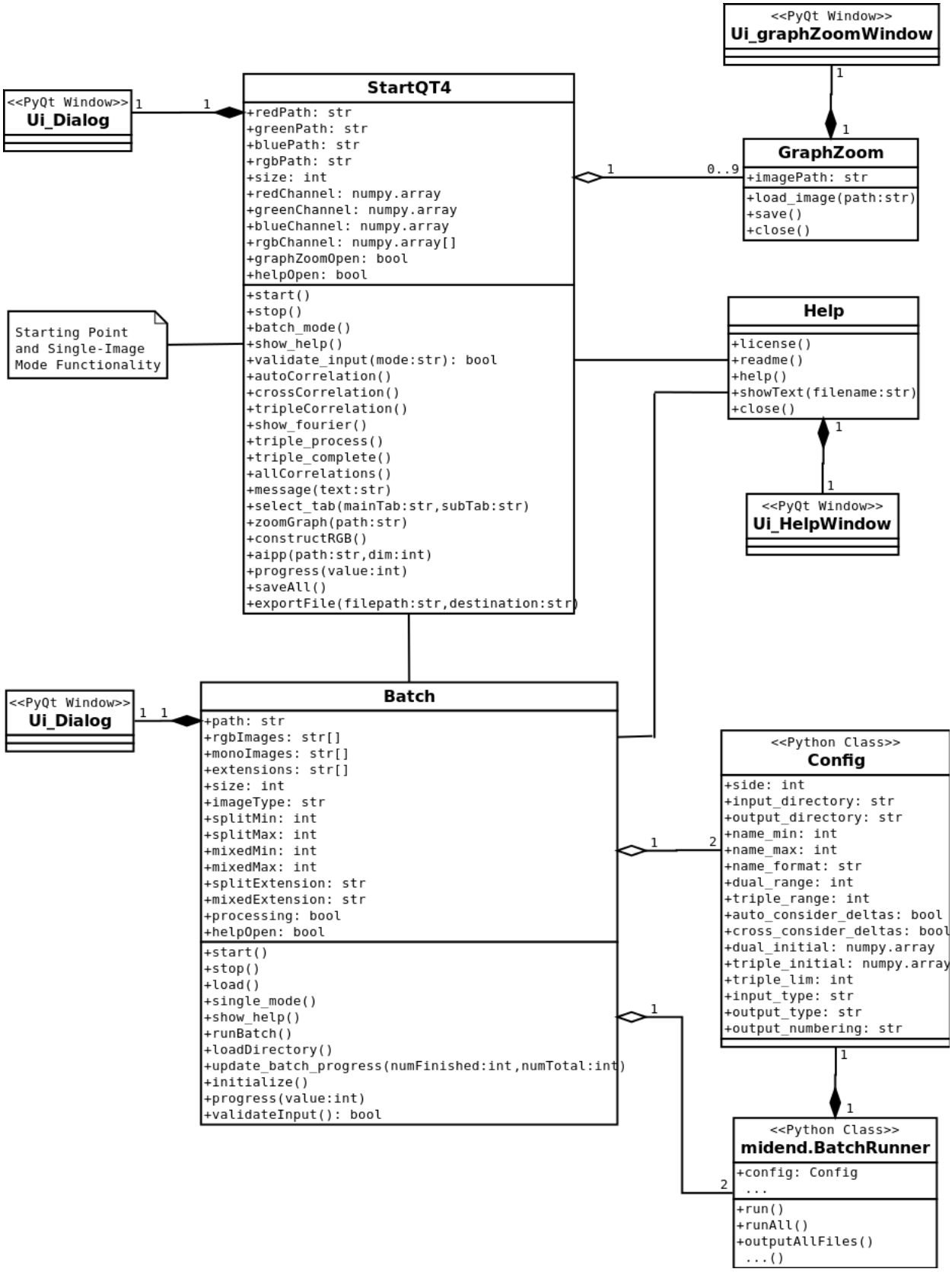
Midend Class Diagram



Web Interface Class Diagram



Local GUI Class Diagram

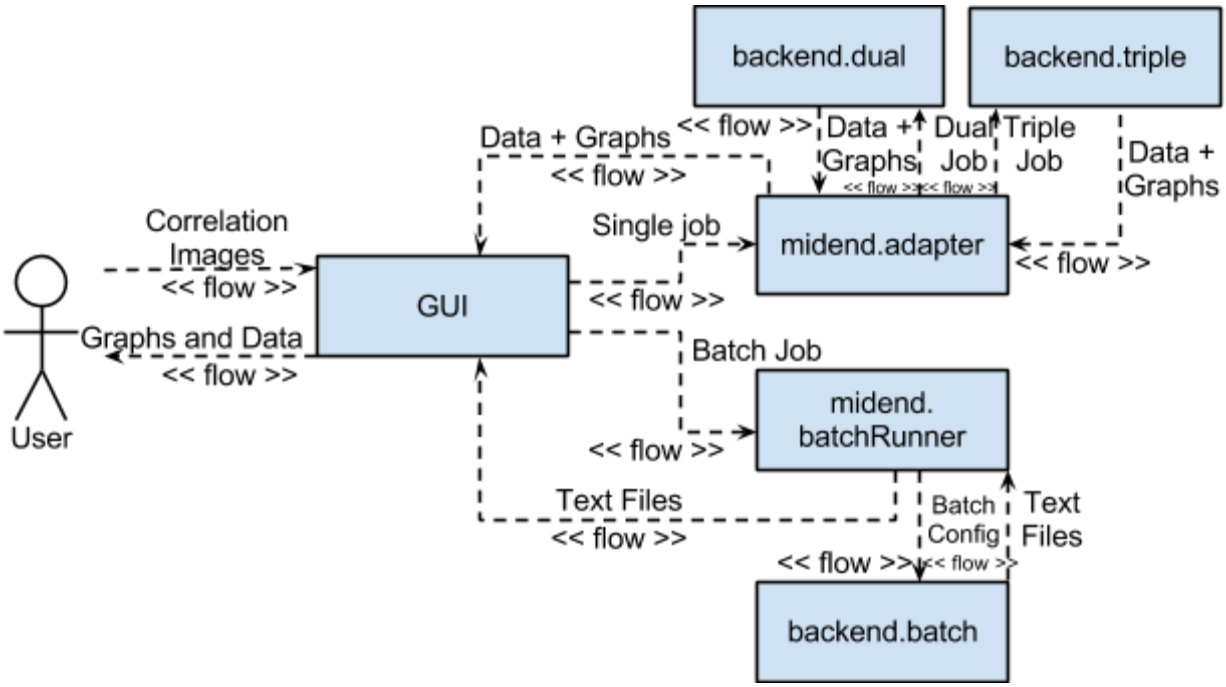


Class Descriptions:

- StartQT4
  - Entry-point class whose init function is run when start.py is called.
  - Provides functionality for single-image mode as well as calls for loading up the Graph Zoomer, Help Window, and Batch Mode.
  - Calls the midend and backend when necessary.
- Ui_Dialog (for StartQT4)
  - Contains all GUI elements for single-image mode.
  - Automatically generated from PyQt Designer UI file.
- Batch
  - Provides functionality for running batch correlations.
  - Calls the midend and backend when necessary.
  - Can recall single-image mode or call Help Window.
  - Does correlations by creating a Config object and sending it to BatchRunner.
- Ui_Dialog (for Batch)
  - Contains all GUI elements for batch mode.
  - Automatically generated from PyQt Designer UI file.
- GraphZoom
  - Called whenever the user clicks the Zoom button on a graph in single-image mode.
  - Contains functionality for displaying the graph, saving as a file, or closing the window.
  - Multiple instances for different graphs can be open at the same time.
- Ui_graphZoomWindow
  - Contains all GUI elements for Graph Zoomer.
  - Automatically generated from PyQt Designer UI file.
- Help
  - Provides functionality for the Help/About dialog
  - Called when the user clicks Help/About in either correlation mode
  - Acts as a text file viewer for HELP, LICENSE, and README.md
  - Converts the first few lines of the HELP file to dialog headings
  - Only one instance is allowed at a time
- Ui_HelpWindow
  - Contains all GUI elements for Help/About Dialog
  - Automatically generated from PyQt Designer UI file.
- Config
  - Object containing all configuration parameters for running batch correlations.
  - Two are made; one for monochrome images and one for mixed images.
  - Sent to the BatchRunner object to actually run the correlations.
- midend.BatchRunner
  - Provides functionality for carrying out batch correlations and outputting results files.
  - Needs a Config object in order to be run.

Dynamic Behaviors

Information Flow



The Information flow of ICS is similar between both the Web GUI and Local GUI. The above diagram abstracts this information.

The User passes in images (and some parameters) to the GUI. The GUI sends it to the midend.adaptor if it is a single job, or the midend.batchRunner if it is a batch job. Single jobs are then divided into multiple dual and/or triple jobs, which go the respective backend process. These processes return result data and graphs, which are then passed back through the chain to the GUI, and then to the User.

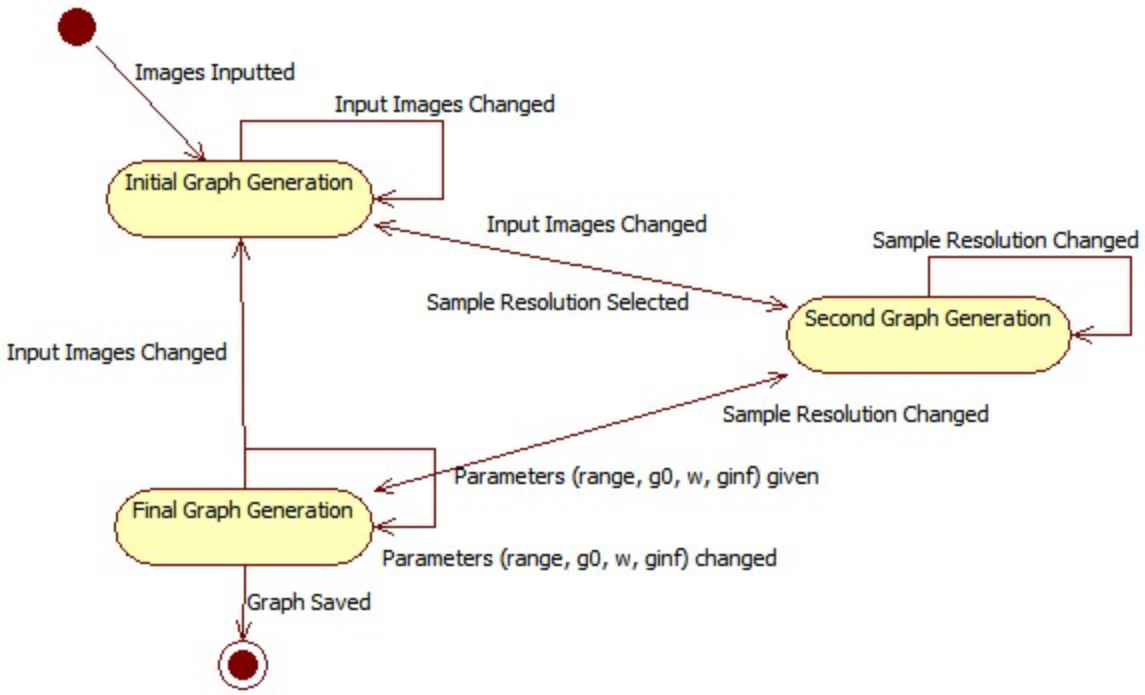
Batch jobs go to the backend batch module, which returns text files, which are given to the user as outputted data.

This conveys the basic flow of the program, and the most basic metaphor: The user gives in images of some type of job, and gets out corresponding data and graphs.

Triple Correlation Activity Diagram

The Triple correlation requires multiple steps from the user in order to create the correct final data. First, the user must add images, which gives the initial graph (the Fourier Transform (Red) Surface Plot). From there, the User may select the Sample Resolution, which determines how much of the initial graph will be used to generate the second graph (Triple-Correlation Surface Plot). From there, final parameters describing the initial guess of the fitting curve (range, g(0), w, ginf) are given.

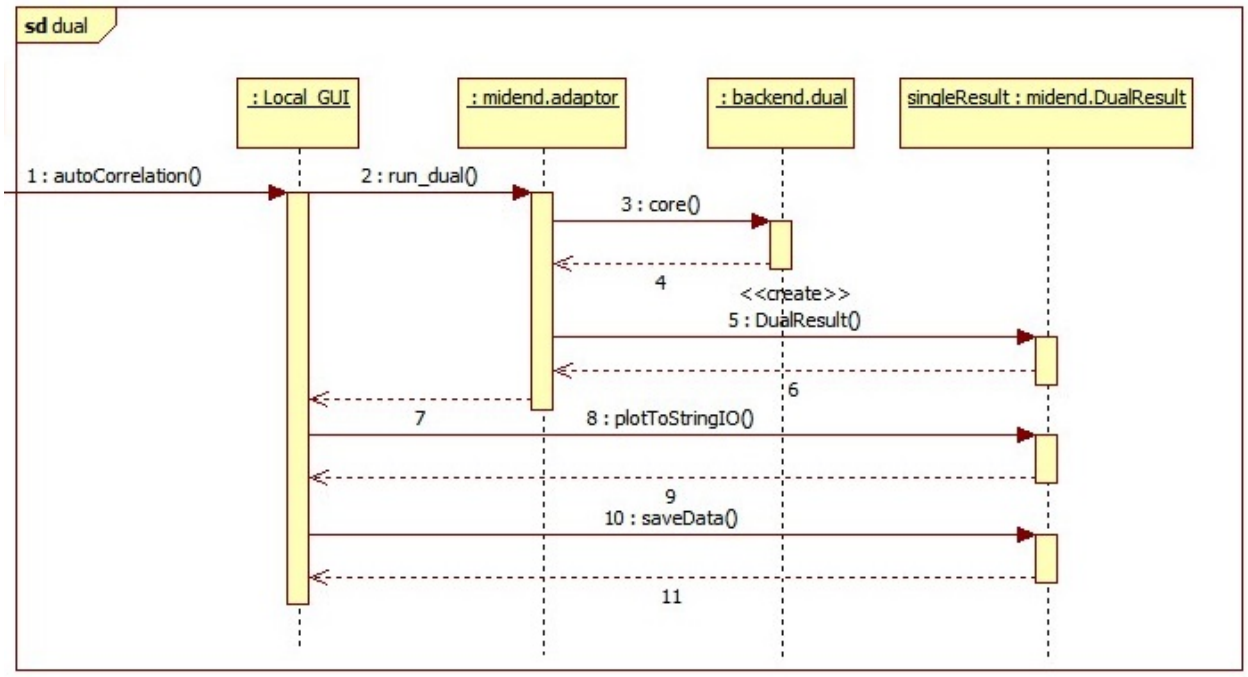
The final graph (the fitting curve) is then generated, and the user may then save the results. However, between each step, the User may always replace the images, choose a different resolution, etc, causing the program to “rewind” to a previous step.



Dual Correlation Sequence

There are three Sequences of note that need to be explained: Dual Correlations, Triple Correlations, and Batch Correlations.

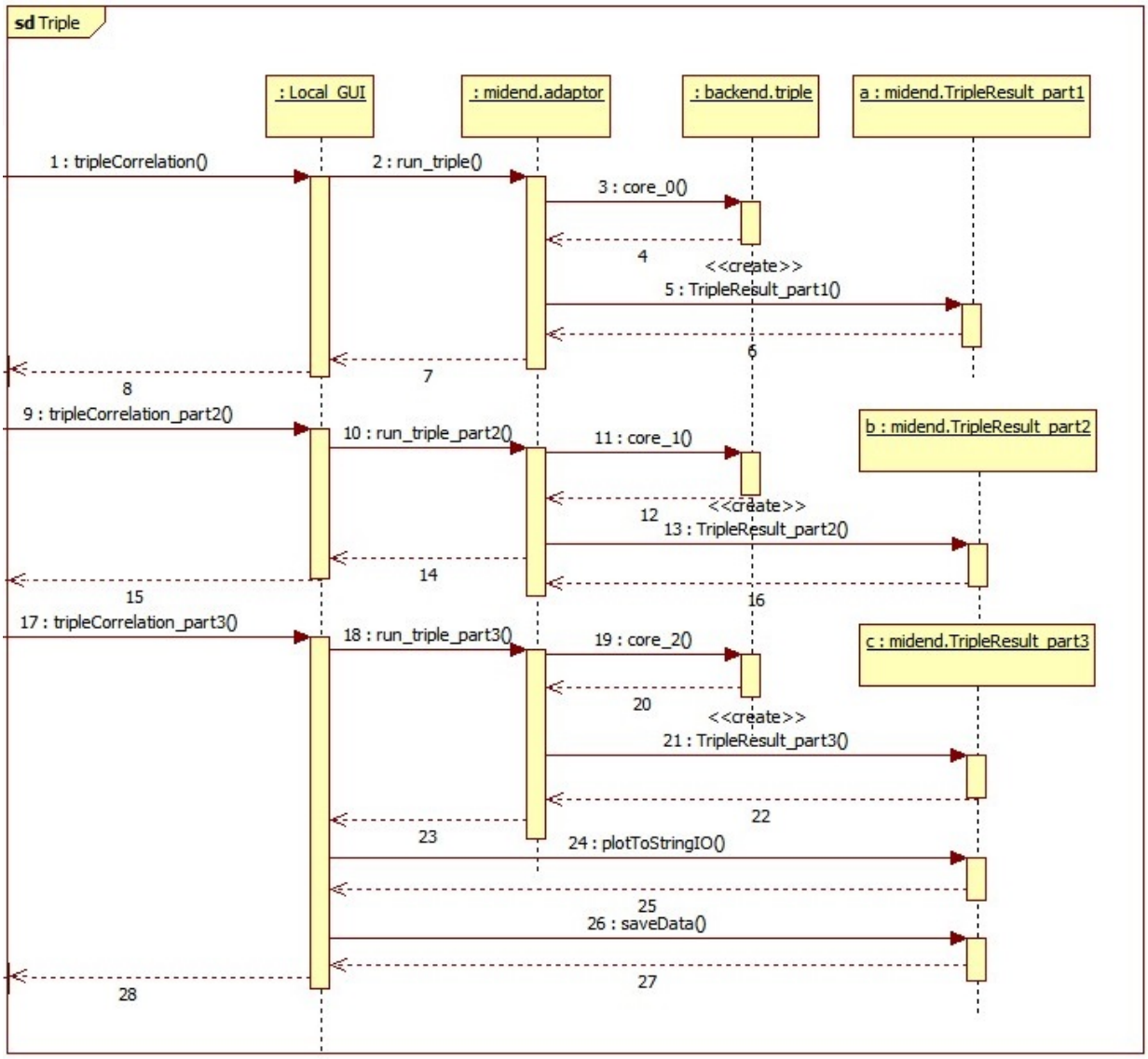
Dual Correlations pass through the GUI to the midend and on into the backend synchronously. It is a relatively simple process, but forms the basis for more complicated processes. The below sequence skips a subcall to a graph module, which was removed to make it fit. It occurs as part of the plotToStringIO call. In addition, there are a number of “run_dual” initial calls to the midend based on whether the image has all three channels or three single channel images are being supplied, etc. This was abstracted away as it doesn’t have a noticeable effect on how the program runs, simply on what is passed in.



The Web GUI uses the same sequence.

Triple Correlation Sequence

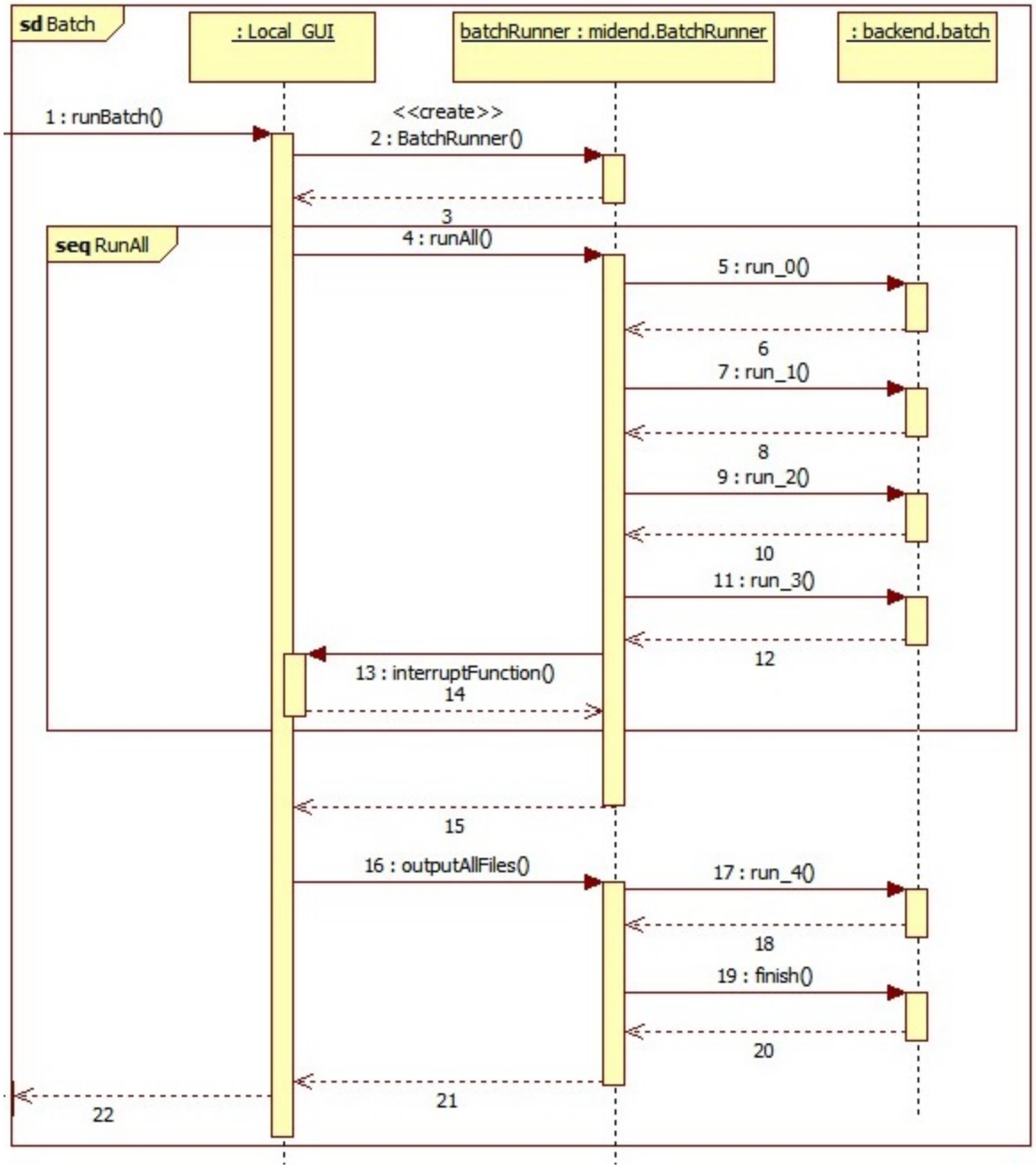
The Triple Correlation Sequence is more complex, but very similar to the dual. As indicated above by the activity diagram, the call sequence requires a large amount of user input. However, the call process is fundamentally the same. Similar assumptions apply as per the Dual Correlation Sequence. Also, not shown here, is the triple correlation calling compiled C code during core_1, which does the expensive sections of the work.



The Web GUI uses the same sequence, with a few additional callbacks to display intermediate webpages.

Batch Correlation Sequence

The final important sequence, The Batch Correlation Sequence is relatively simple. It is basically made up of a number of “run” commands in the backend that are encapsulated by the midend. This sequence diagram uses the “runAll” command, which uses callbacks between each run. There is another version which instead returns control between each run. Also, not shown here, the backend calls on compiled C code during run_2. This is where the triple correlation is done.



The Web GUI uses the same sequence, with the only notable change is it does not make use of the interrupt callback.