CARTA Interface Control Document

Angus Comrie and Rob Simmonds

May 05, 2022

CONTENTS:

Date 05 May 2022

Authors Angus Comrie, Rob Simmonds and the CARTA development team

Version 27.2.0

ICD Version Integer 27

CARTA Target Version 3.0

CHAPTER

ONE

CHANGELOG

Version	Date	Description
0.1.7	30/08/18	Added optional field `channel_histogram_data` to the image view message
0.1.8	07/09/18	Added `computed_entries` map to the File Info Extended message
0.1.9	10/09/18	Changed `computed_entries` map to an array of *HeaderEntry* messages
0.1.10	10/09/18	Added optional field `spatial_requirements` to the *SET_CURSOR* message
0.1.11	20/09/18	Added `file_id` field to *SET_REGION* and fixed control points typo
0.1.12	18/10/18	Deprecated `channel_vals` field in *SPECTRAL_PROFILE_DATA*
0.1.13	30/11/18	Added details on per-cube histogram calculations
0.1.14	07/02/19	Added `$BASE` folder placeholder
0.1.15	05/03/19	Added target version info
0.1.16	25/03/19	Removed `stokes`, `channel_min` and `channel_max` fields from *SET_REGION*, and changed *StatisticsValue* `value` field to a `double` type.
0.1.17	27/03/19	Added `NumPixels` and `NanCount` stats types
0.1.18	28/03/19	Changed rotation units from radians to degrees
0.2.0	07/05/19	Changed message header information, adjusted stats enum values, added double value support for spectral profile
0.2.1	09/05/19	Added feature flag enums as well as event type enums. Updated animation information and sequence diagrams to include flow control
0.2.2	14/05/19	Added information on tiled rendering
4.0.0	02/07/19	Expanded sequence diagrams and text on tiled rendering and animation. Changed version numbering to match ICD version integer
4.0.1	04/07/19	Fixed incorrect sequence diagrams for file loading
5.0.0	15/07/19	Switched to byte fields instead of repeated float/double for efficiency reasons in spatial and spectral profile messages
6.0.0	19/07/19	Animation ID and timestamps for ACKs
7.0.0	23/07/19	Region file browser and import/export messages
7.0.1	08/08/19	Region export coordinate type
8.0.0	21/08/19	Added messages for retrieving and setting user preferences and layouts
9.0.0	17/10/19	Added/updated messages for contour parameters and streaming
10.0.0	25/10/19	Updated messages for contour streaming
11.0.0	20/11/19	Added messages to resume the session and its ACK
12.0.0	18/02/20	Updated messages for tiled rendering usage during animation
13.0.0	19/05/20	Updated messages for scripting service information (WIP)
14.0.0	28/05/20	Updated messages for catalogs
15.0.0	04/07/20	Added date field to file info, reordered file types alphabetically
16.0.0	23/07/20	Added sub-message for region style, replace `RegionProperties` with map
17.0	27/07/20	Added spectral line request and response

continues on next page

Table 1 – continued from previous page

Version	Date	Description
17.1.0	11/08/20	Non-breaking change: added map of matched frames for spectral matched animation
17.2.0	12/08/20	Non-breaking change: added intensity limit field to line ID query
18.0.0	11/12/20	Added `extrema` enum value to *StatsType*. Removed deprecated messages: `SetUserLayout`, `SetUserLayoutAck`, `SetUserPreferences`, `SetUserPreferencesAck`, and `SetRegionRequirements`.
18.1.0	08/01/21	Non-breaking change: added `beam_table` (of type *Beam*) to *OpenFileAck*.
19.0.0	07/01/21	Adjusted `FileInfoResponse` to include map of extended file info messages
20.0.0	13/04/21	Added `ConcatStokesFiles` messages
20.1.0	23/04/21	Renamed `REGION_WRITE_ACCESS` to `READ_ONLY` in *ServerFeatureFlags*.
20.2.0	26/04/21	Added additional fields to *SaveFile* for sub-image support.
21.0.0	05/05/21	Added *ListProgress* and *StopFileList* messages
22.0.0	28/06/21	Added *DirectoryInfo* message.
23.0.0	28/06/21	Added *SpatialConfig* submessage with fields for spatial profile mip and range, and added mip field to *SpatialProfile*. Updated comments for *SpatialConfig* and *SpectralConfig*.
23.1.0	23/07/21	Added *SplataloguePing* and *SplataloguePong* messages.
23.1.1	29/07/21	Added `return_path` to *ScriptingRequest* message.
24.0.0	30/07/21	Added the stokes to *SetStatsRequirements*, *HistogramConfig*, and *RegionHistogramData* messages. Removed the channel from *Histogram*.
24.1.0	12/10/21	Added `platform_strings` to *RegisterViewerAck* message.
24.2.0	11/11/21	Added `filter_mode` to *FileListRequest*, *CatalogListRequest* and *RegionListRequest* messages; Added `Unknown` to *CatalogFileType*.
25.0.0	06/12/21	Added *PvRequest*, *PvResponse*, *PvProgress*, and *StopPvCalc* messages for PV generator.
26.0.0	13/01/22	Removed `grpc_port` from *RegisterViewerAck* message. Renamed `GRPC_SCRIPTING` to `SCRIPTING` in *ServerFeatureFlags*. Removed all references to gRPC in docs.
26.1.0	01/03/22	Added `lel_expr` to *OpenFile* message.
26.2.0	19/04/22	Added `rest_freq` to *SaveFile* message.
27.0.0	21/04/22	Added *FittingRequest* and *FittingResponse* messages for image fitting.
27.1.0	27/04/22	Added `Ptotal`, `Plinear`, `PFtotal`, `PFlinear`, and `Pangle` to *PolarizationType* enum. Added `stokes_indices` to *StartAnimation* message.
27.2.0	05/05/22	Added *SetVectorOverlayParameters* and *VectorOverlayTileData* messages. Moved *TileData* to `shared`

CHAPTER

TWO

VERSIONING

- Major version change (`1.2.3 -> 2.0.0`): this is a breaking change.
- Minor version change (`1.2.3 -> 1.3.0`): this is added functionality which is optional and non-breaking.
- Patch (`1.2.3 -> 1.2.4`): this is a change which does not affect functionality (e.g. a typo fix in a comment, or a changed field name).

Some legacy changelog entries may not follow this approach. Only changes to the protocol buffer source files should be recorded here; changes only to this documentation do not require a version bump.

2.1 Introduction

The CARTA application is designed in a server-client model, with the backend (written in C++) communicating with the frontend (Web-based, using HTML and JavaScript web frameworks) through an interface defined in this document. While CARTA is required to support a number of file formats (FITS, CASA, HDF5 and Miriad), throughout the document, nomenclature will be defaulted to FITS files, such as when referring to multiple HDUs in a file, and header entries.

Throughout this document, things that require some clarity, or are not finalised are commented on in this font style.

2.2 Context

There are two distinct usage scenarios for frontend-backend configuration in CARTA. Firstly, when used as a desktop application, the frontend and backend both run locally. The backend is run as an application that communicates with the frontend, which is presented to the user as a desktop application in the form of an Electron-wrapped web view [1].

The second usage scenario is that of a remote viewer, where the backend is running on a remote server, while the frontend is loaded in the user's browser of choice (as long as that choice is Chrome, Firefox, Safari or Edge) by visiting a URL associated with the remote server. A third possible configuration is running the desktop Electron application, while connecting to a specific server IP for remote data. This is not a high priority, as most usage scenarios would be better handled through accessing the frontend through a remote URL.

In both of these scenarios, communication between the frontend and backend takes place over a standard WebSocket [2] communication channel, with message formats defined using protocol buffers [3], based on the message structures defined in *Section 4.1*.

Image data is sent to the frontend as either uncompressed or compressed floating point data. The frontend can request which type of data is sent from the backend, which compression library to use, and what compression quality to use. Two lossy floating-point compression libraries are supported in the ICD: ZFP [4] and SZ [5] (although SZ is not implemented at this point on either the frontend or backend). A general investigation of the compression performance of these two libraries shows that ZFP is consistently faster, while SZ offers slightly better compression ratios at the expense of

compression and decompression speed. The current implementation of the SZ library is not thread safe, meaning that compression on the backend would have to be implemented sequentially. Note that, due to the frontend's use of web workers to decompress data, this limitation is overcome, as each web worker operates in a separate execution space. ZFP should be preferred when network bandwidth is sufficient. In the case of a desktop application, uncompressed data or very high quality ZFP compressed data should be favoured. When using uncompressed data, the FP32 floating point data is copied directly to and from the uint8 array specified by *TileData* (using 4 uint8 entries per 32-bit floating point entry).

Contour data is streamed as either uncompressed floating points, or compressed decimated fixed-point data. Contour data is losslessly compressed using the Zstandard [6] library, after being decimated to a fixed-point value. Vector overlay data follows the same approach as image data.

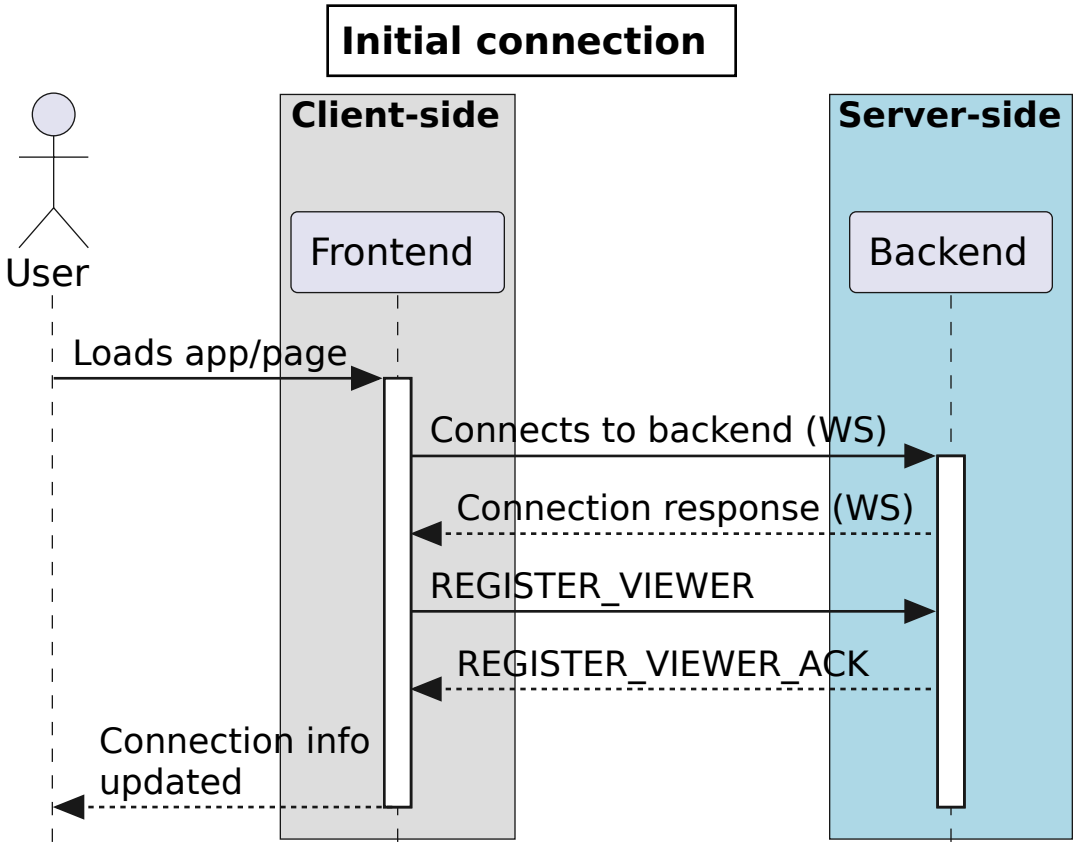
2.3 Behaviour

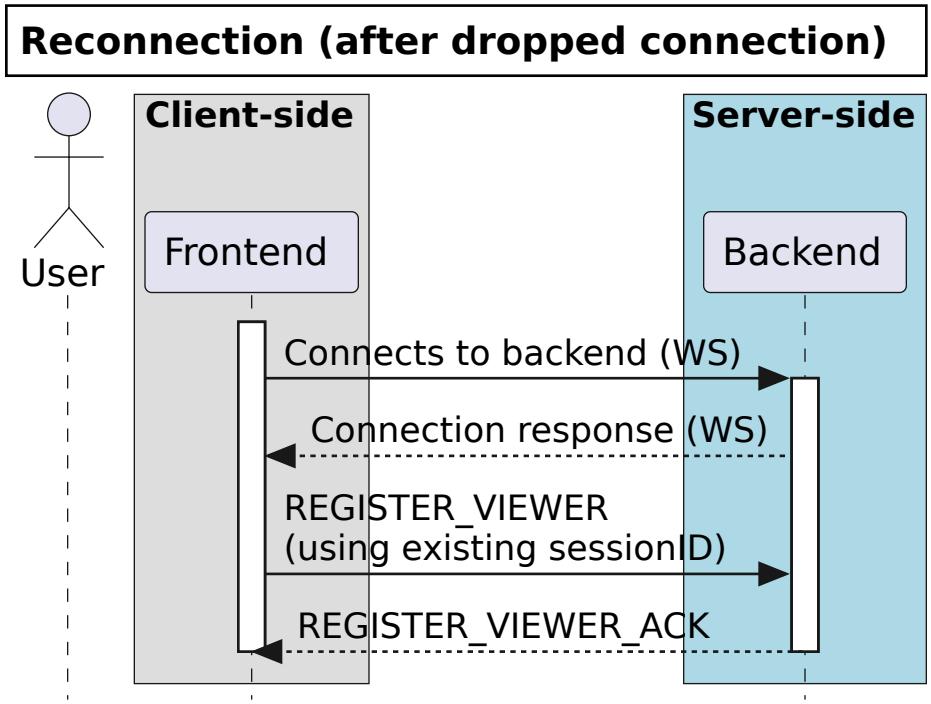
2.3.1 Connection

Connection takes place via the WebSockets protocol, and is initiated as soon as the frontend page is successfully loaded. Upon connection, the frontend registers itself to the backend using the *REGISTER_VIEWER* message and retrieves a new session ID, server capabilities and user preferences through *REGISTER_VIEWER_ACK*. It then requests the list of files in the default directory. If the connection is dropped, the frontend re-registers itself to the server, but passes through the original session ID. The server should attempt to resume this session, but if not possible, will generate a new session ID for the client. In addition to the session ID, the frontend can pass through an optional API key, which can be used to determine basic permissions and user-related settings.

A connection heartbeat is established by the server-initiated ping/pong sequence defined by the WebSocket protocol. In addition to this, a client-initiated ping/pong sequence is produced by empty messages being sent by the frontend periodically. The backend keeps track of the time since each connected client last initiated the ping/pong sequence, and makes timeout decisions based on this value.

When the frontend is intentionally closed, by closing the associated app or web page, the frontend closes the WebSocket connection gracefully, and the backend can then remove the associated session. When the frontend is closed in error, or the backend determines that a connection is timed out, the backend should maintain the session for an appropriate period, so that it can be resumed when the frontend reconnects. The frontend should attempt to reconnect with the same session ID when a connection is dropped. If the backend responds with a session type set to `RESUMED`, the frontend will attempt to resume the session by sending a list of files, along with their associated regions in a *RESUME_SESSION* message.



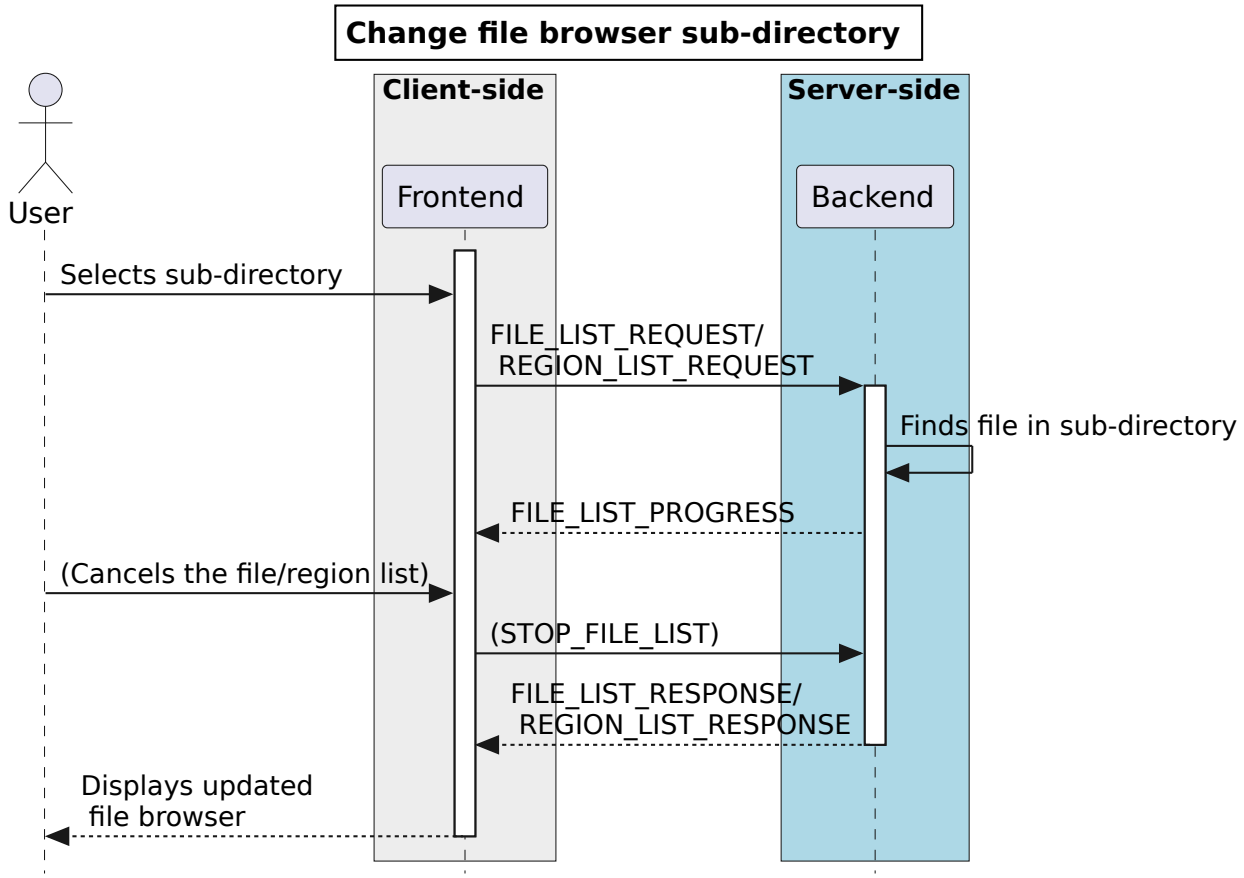


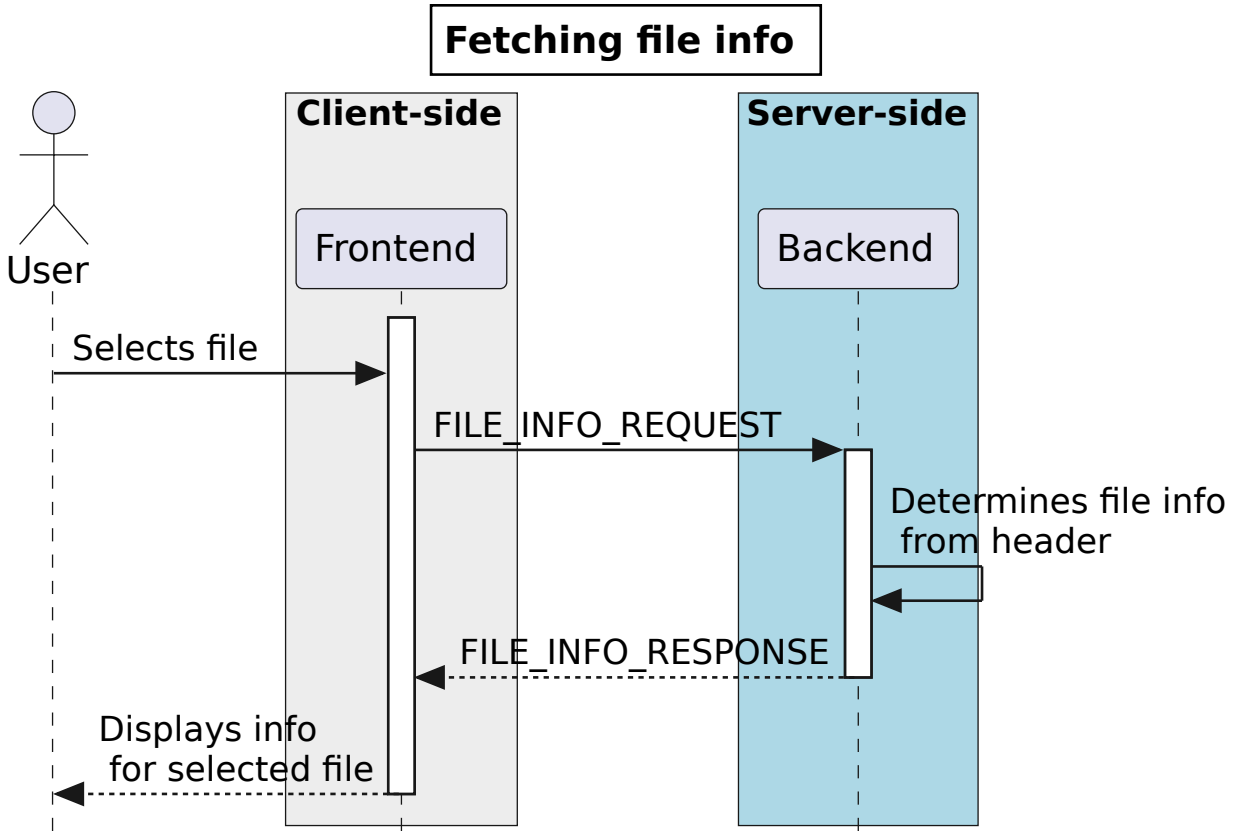
If the scripting interface is enabled, the backend HTTP server accepts scripting requests, acting as a proxy be-

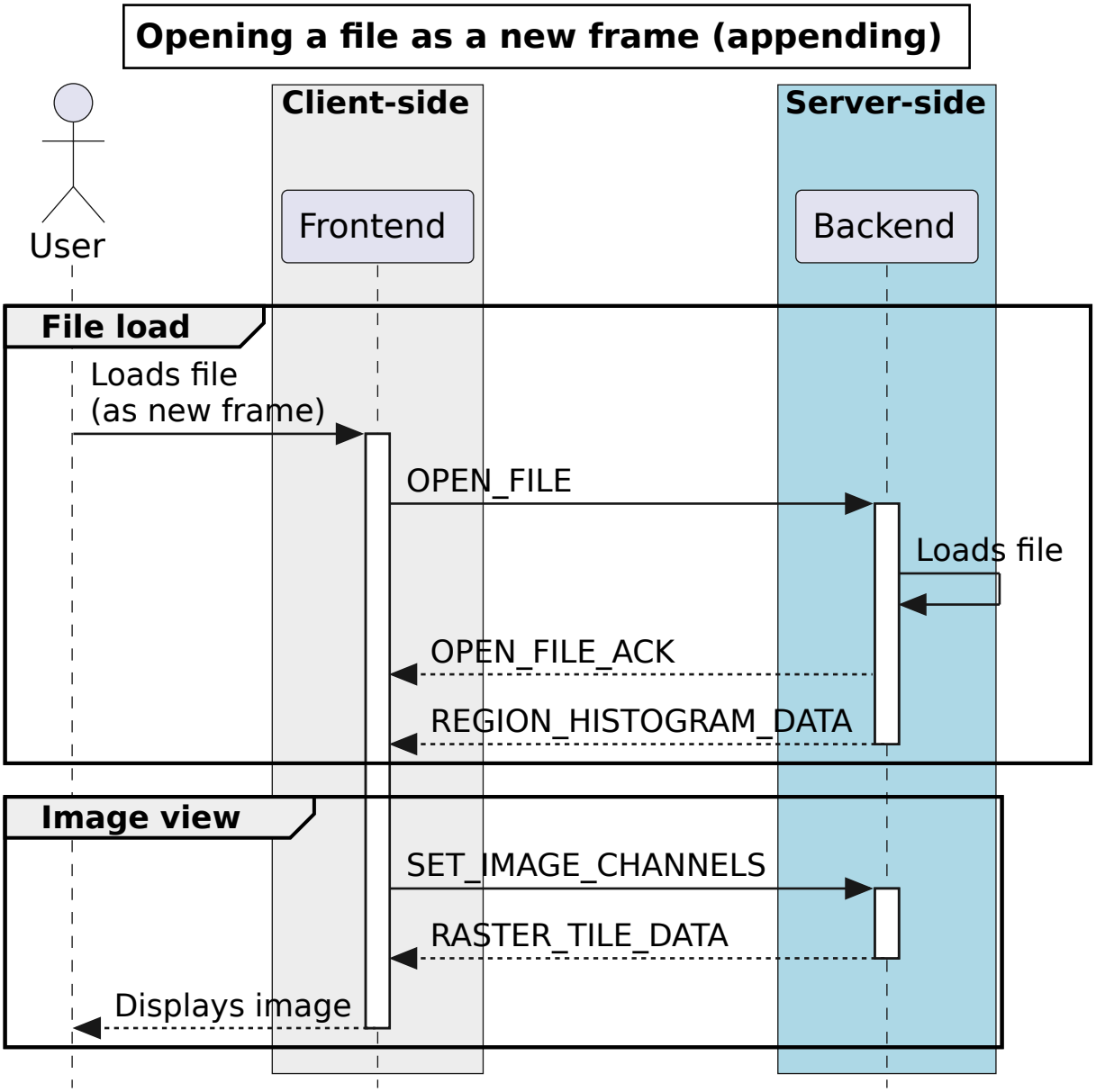
tween a scripting client, such as a python package, and the frontend. The frontend parses a scripting command from each `SCRIPTING_REQUEST` message sent by the backend, executes the required code, and responds with a `SCRIPTING_RESPONSE` message, which includes the success state of the command, as well as a possible response in JSON format. Each incoming scripting request includes a unique ID, which is passed back in the scripting response, so that the backend can uniquely match scripting requests to their responses.

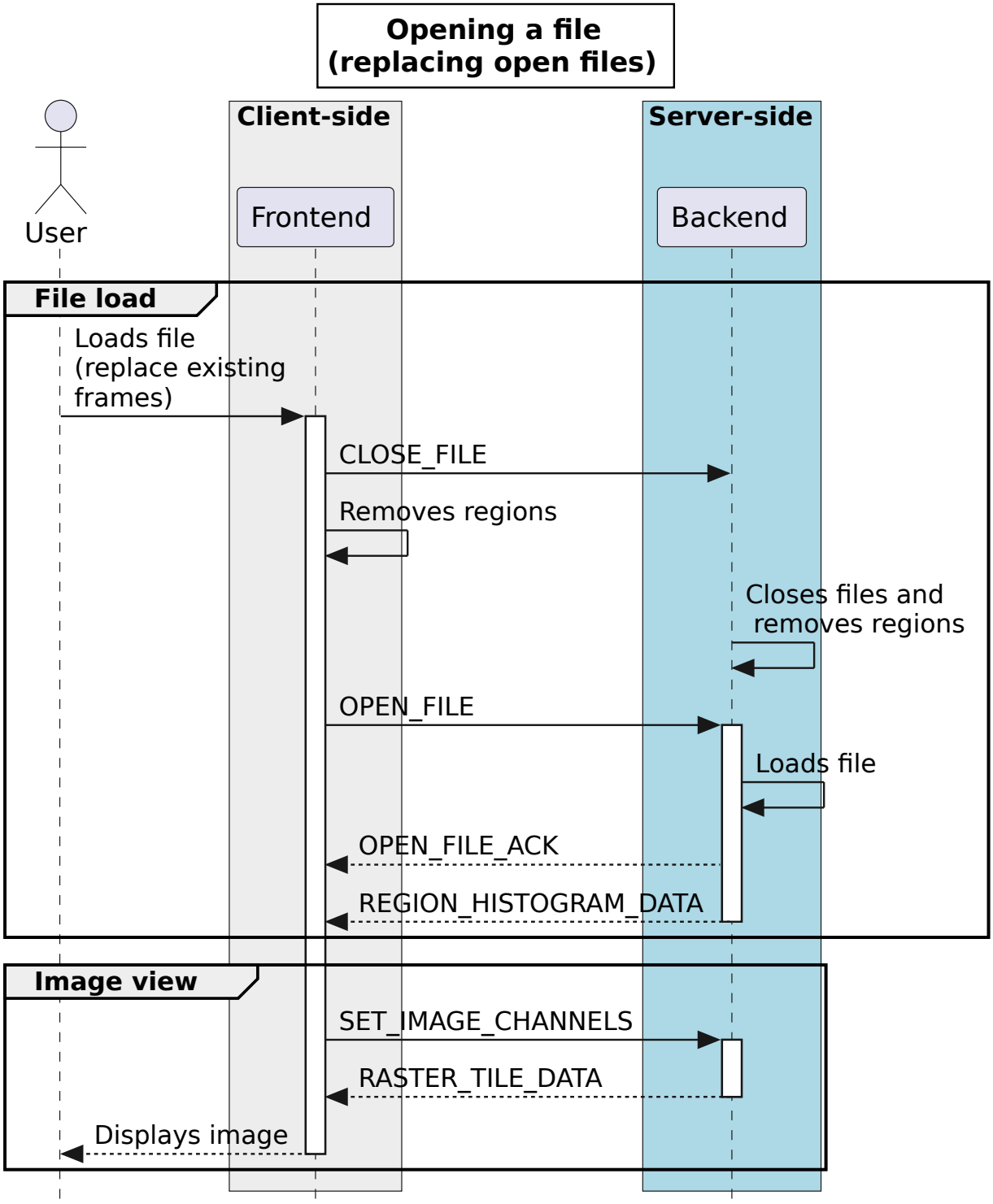
2.3.2 File browsing

The file browser displays a list of files in the selected directory, along with some basic information on each file (type, size) and a list of subdirectories. If a file contains multiple HDUs (or equivalent), a list of HDU names is included. If a file is selected in the file browser, additional information is shown. A specific HDU of a file can be selected. When a subdirectory is selected, the file list is fetched for that subdirectory. When a file is loaded, the default image view is requested. A file can be loaded as a raster or contour image (not currently implemented), and can be appended to the current list of open files, or can replace all open files, in which case the frontend must first close all files using the *`CLOSE_FILE`* message with `file_id = -1`. Individual open files can be removed from the file list by calling *`CLOSE_FILE`* with an appropriate `file_id` field.




Fetching file info
Client-side
Server-side
User
Frontend
Backend
Selects file
FILE_INFO_REQUEST
Determines file info
from header
FILE_INFO_RESPONSE
Displays info
for selected file


Opening a file as a new frame (appending)
Client-side
Server-side
User
Frontend
Backend
File load
Loads file
(as new frame)
OPEN_FILE
Loads file
OPEN_FILE_ACK
REGION_HISTOGRAM_DATA
Image view
SET_IMAGE_CHANNELS
RASTER_TILE_DATA
Displays image


Opening a file
(replacing open files)
User
Client-side
Frontend
Server-side
Backend
File load
Loads file
(replace existing
frames)
CLOSE_FILE
Removes regions
Closes files and
removes regions
OPEN_FILE
Loads file
OPEN_FILE_ACK
REGION_HISTOGRAM_DATA
Image view
SET_IMAGE_CHANNELS
RASTER_TILE_DATA
Displays image

2.3.3 Data cube navigation

The frontend can change the displayed channel and Stokes parameter by issuing the *SET_IMAGE_CHANNELS* command. When an image is opened, the frontend will send a *SET_IMAGE_CHANNELS* with the first channel and Stokes parameter. The frontend subscribes to all *RASTER_TILE_DATA* messages.

Tiled rendering splits the image into individual square tiles (defaulting to 256 pixels in width), and renders the image progressively as tiles arrive from the backend. This is more efficient when exploring a large image, as it reuses data when panning and zooming around the image. Images are downsampled by a power of 2.

In addition, contour rendering can be used on files. The contours for an entire channel are generated when the frontend sends the *SET_CONTOUR_PARAMETERS* command. The frontend subscribes to all *CONTOUR_IMAGE_DATA* messages. Currently, contour renders are automatically updated when the user changes channel or plays an animation. Contours are delivered in separate chunks by the backend, so that the user can see the contours as they are delivered to the frontend, and can get an idea of how long the contour fetching will take.

Zooming and panning

The frontend can request specific tiles of an image to be delivered. Tiles are specified using the widely used a tiled web map convention (commonly used in GIS and online image viewer software). Each tile is defined by three coordinates: The layer, x and y coordinates. The zeroth layer consists of the entire image, down-sampled until it is stored in a single tile, with both width and height less than or equal to a chosen tile size (defaulting to 256 pixels, but this may increase in future to 512 pixels for large format screens). The tile size must be a multiple of four, due to the ZFP algorithm's block size. Each subsequent layer doubles in width and height, to the point where the highest layer (N) contains the entire image in full resolution, split into fixed-size tiles (tiles along the right and top edges of the image will have reduced width and height respectively).

Tile coordinates (`layer`, `x` and `y`) are encoded into a single 32-bit integer before sending. There are two primary reasons for this:

- Using a struct as a key in a map on either frontend or backend would be more complicated, and require a custom hash function. JavaScript `Map` objects do not support this. Storing tiles within a map-of-maps-of-maps would be less efficient.
- Encoding and decoding an array of structs in a protocol buffer object would be less efficient in terms of CPU time and network storage

The encoded integer consists of:

- 12 bits for the X and Y coordinate. This limits the implementation to at most 4096 tiles along either axis. With a default tile size of 256 pixels, this means images must be smaller than 1.04 million pixels in width and height.
- 7 bits for the layer coordinate. This limits the implementation to 128 layers. However, this limitation is artificial, since at most 12 layers will be required, given the above limitation of 4096 tiles
- 1 bit left over, because JavaScript bit shifting is done on signed integers, rather than unsigned

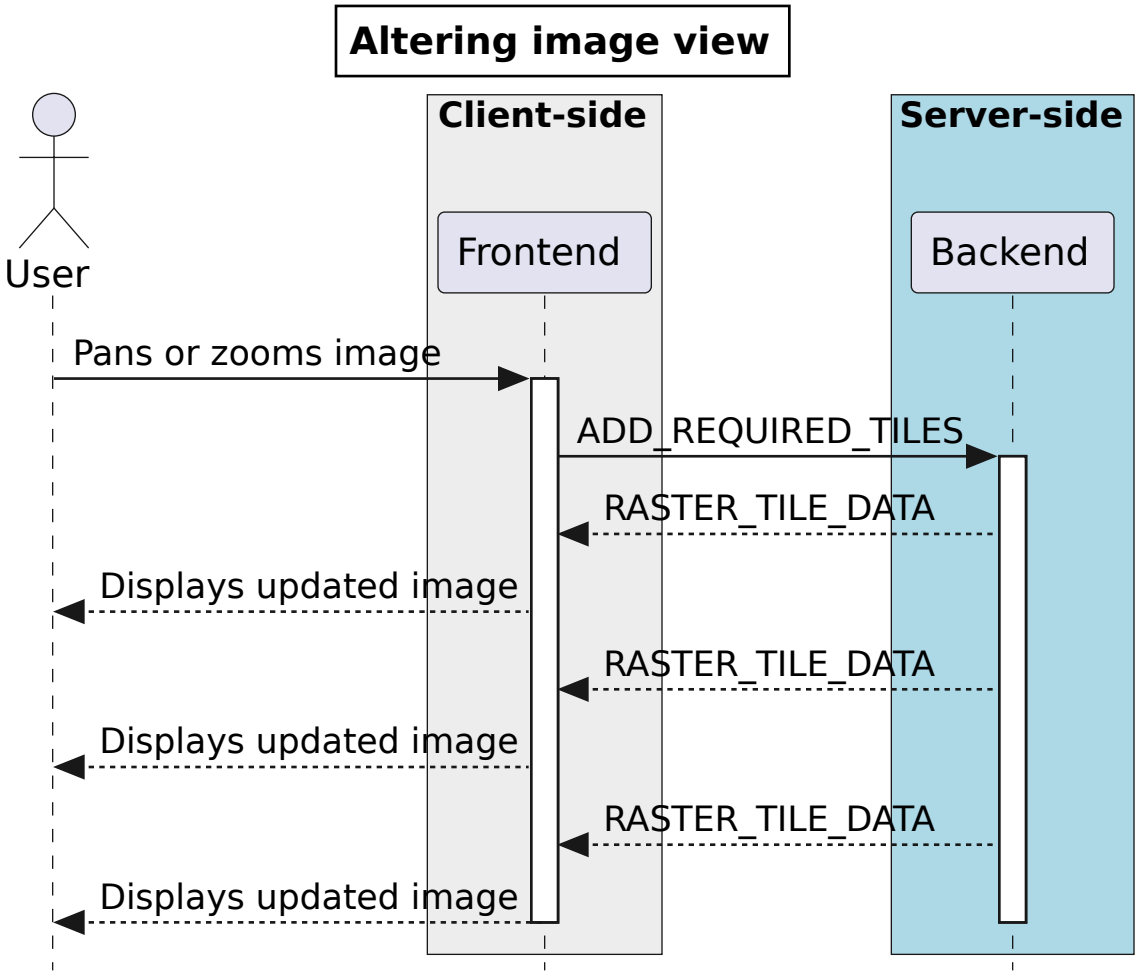
Encoding and decoding is a simple and lightweight process using some bit shifting. A single line JavaScript function to encode is:

```
(x, y, layer) => (layer << 24) | (y << 12) | x;
```

When a user zooms or pans, the frontend sends the *ADD_REQUIRED_TILES* command to the backend. The frontend may debounce, throttle or delay sending tiles to the backend, in order to optimise delivery and avoid sending stale tiles. The order of the list of tiles supplied to *ADD_REQUIRED_TILES* determines the order in which the backend delivers tiles. If subsequent *ADD_REQUIRED_TILES* messages arrive while the backend is still delivering tiles, the most recent tile list is prioritised.

Another route for optimisation available to the frontend is *REMOVE_REQUIRED_TILES*, which allows the frontend to explicitly indicate that certain tiles are no longer required. If any of these tiles are yet to be delivered to the frontend, the backend can optimise tile delivery by removing them from the queue of titles to be delivered.

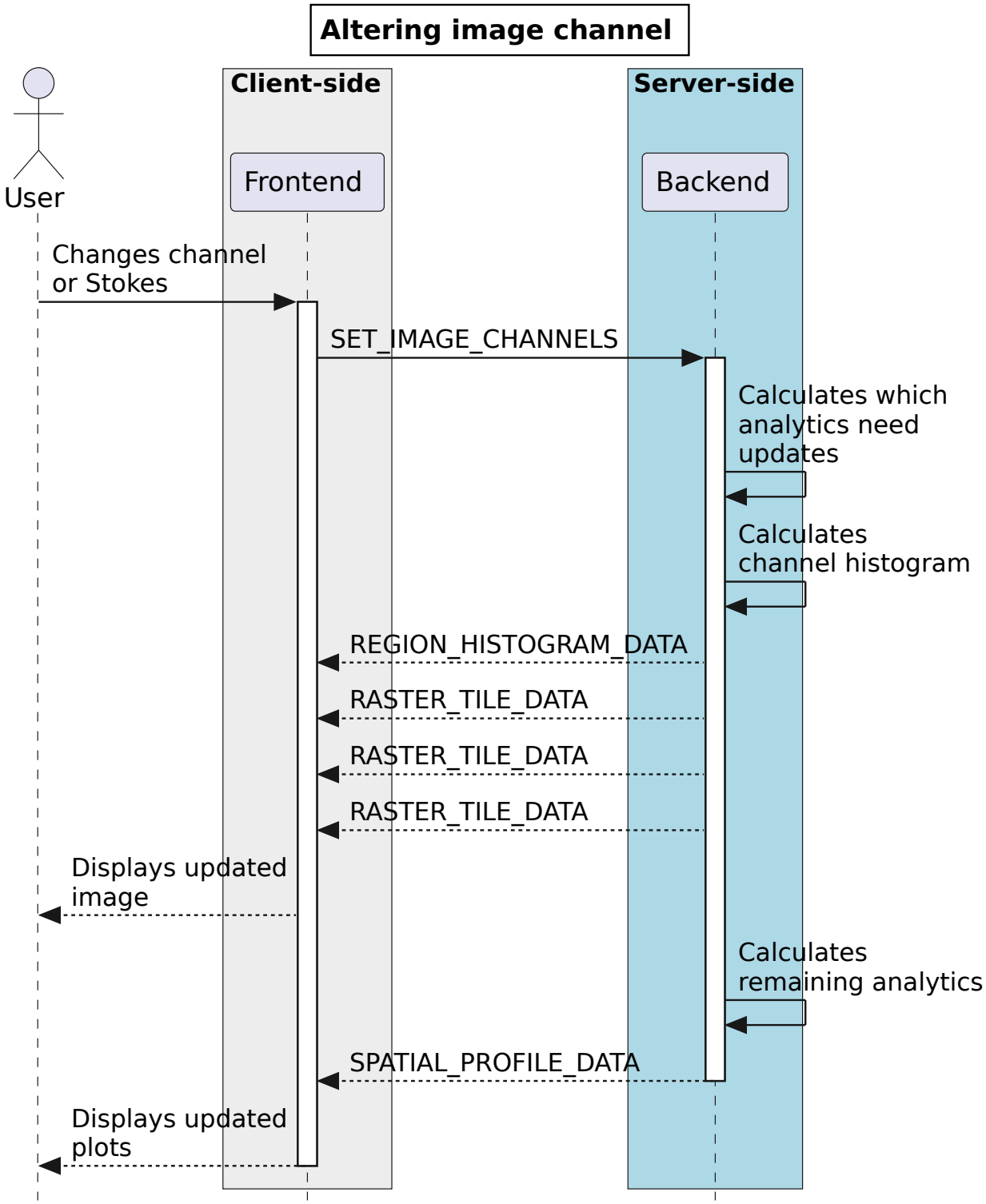
Tile data is delivered by the backend using the *RASTER_TILE_DATA* stream. This allows the backend to send one or more raster tiles with the same compression format and quality to the frontend. Each time a tile is delivered to the frontend, the image is re-rendered.



Channel navigation

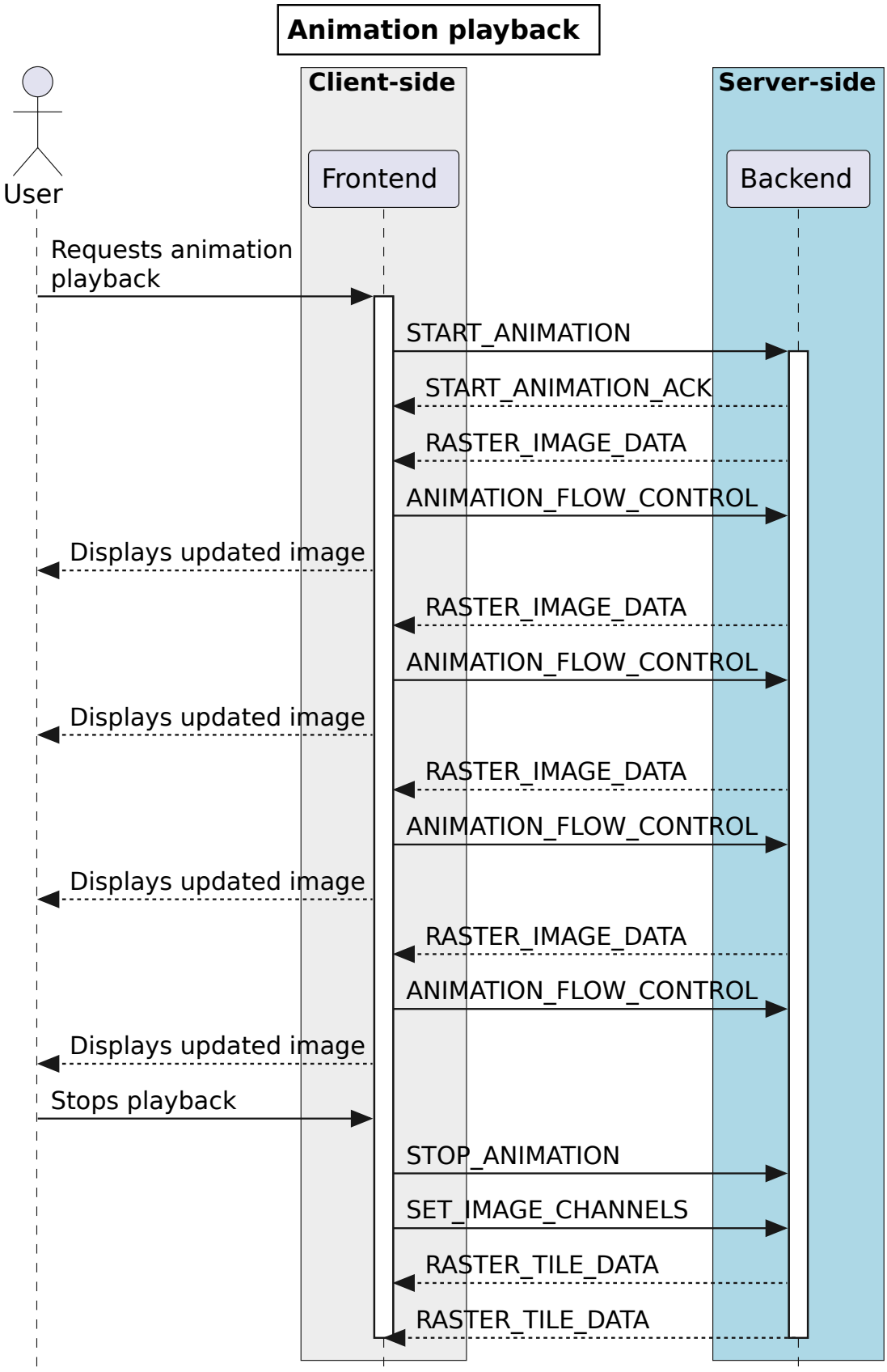
When changing channels via a *SET_IMAGE_CHANNELS* message, the frontend includes an initial list of required tiles. These tiles are then delivered individually by the backend. Unlike the case when zooming and panning, the frontend will wait for all required tiles to be delivered before displaying an image when switching channels. When receiving a *SET_IMAGE_CHANNELS* message, the backend will also send the new channel histogram via the *REGION_HISTOGRAM_DATA* stream.

In general, one image view command will correspond to a subsequent image data stream message. However, changing the image channel will result in a subsequent image data stream message, as well as any relevant updated statistics, histograms or profile data.


Altering image channel
Client-side
Server-side
User
Frontend
Backend
Changes channel or Stokes
SET_IMAGE_CHANNELS
Calculates which analytics need updates
Calculates channel histogram
REGION_HISTOGRAM_DATA
RASTER_TILE_DATA
RASTER_TILE_DATA
RASTER_TILE_DATA
Displays updated image
Calculates remaining analytics
SPATIAL_PROFILE_DATA
Displays updated plots

Animation

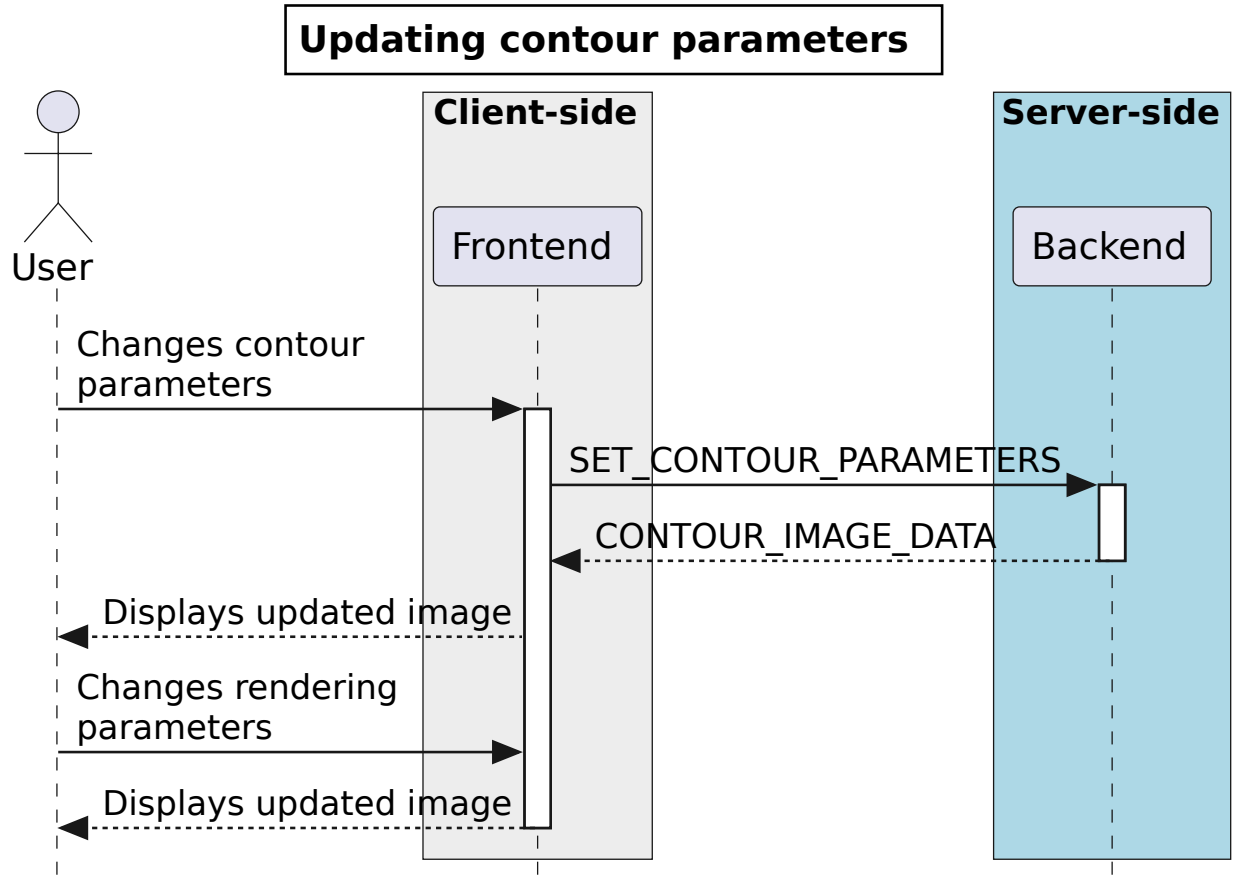
An animation can be played back by issuing the *START_ANIMATION* command. This command encapsulates all the different animation stepping and bounds parameters, in order to allow the backend to perform frame calculations and deliver image data to the front. After the the *START_ANIMATION* command has been issued, the backend sends images and analysis results to the frontend at a regular interval. When the user stops an animation, the frontend sends the *STOP_ANIMATION* command, which includes information on the current image's channels, so that the backend can be sure that the frontend channel state is the same as that of the backend. If the last sent frame does match the frontend channel state, the backend adjusts channels again. In order to prevent the backend from sending too many animation frames, some basic flow control is provided through *ANIMATION_FLOW_CONTROL* message. This is sent from the frontend to the backend to indicate the latest frame received, preventing the backend from queuing up too many frames. The *START_ANIMATION* command includes an *ADD_REQUIRED_TILES* sub-message, specifying the required tiles and compression type to be used in the animation. The backend includes an animation ID field in *START_ANIMATION_ACK* in order to allow the frontend to differentiate between frames of previous animations and the latest animation.


Animation playback
Client-side
Server-side
User
Frontend
Backend
Requests animation playback
START_ANIMATION
START_ANIMATION_ACK
RASTER_IMAGE_DATA
ANIMATION_FLOW_CONTROL
Displays updated image
RASTER_IMAGE_DATA
ANIMATION_FLOW_CONTROL
Displays updated image
RASTER_IMAGE_DATA
ANIMATION_FLOW_CONTROL
Displays updated image
RASTER_IMAGE_DATA
ANIMATION_FLOW_CONTROL
Displays updated image
Stops playback
STOP_ANIMATION
SET_IMAGE_CHANNELS
RASTER_TILE_DATA
RASTER_TILE_DATA

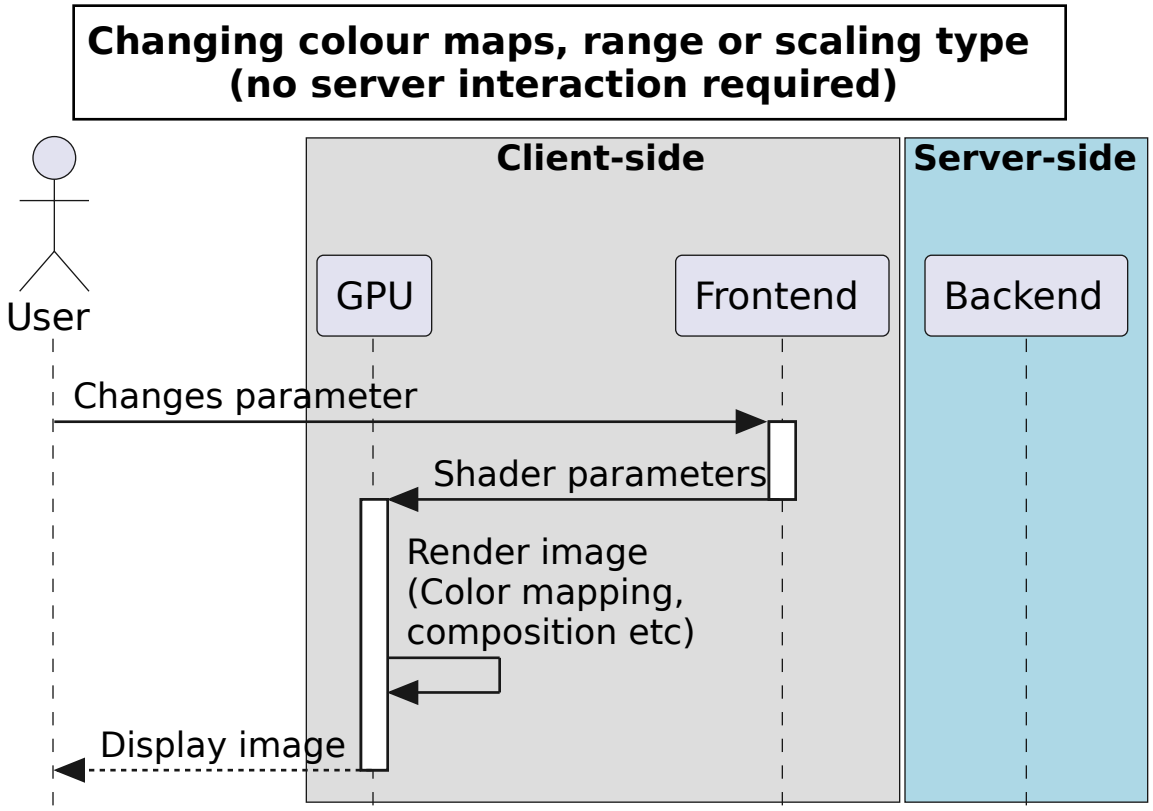
Images are sent as tiled data. In order to keep the image view channel and full image histogram synchronised, the `RASTER_IMAGE_DATA` message includes a *`REGION_HISTOGRAM_DATA`* object, containing the channel histogram for the new channel. During animation playback, each animation step will result in image data stream messages, as well as any relevant analytics updates. If zooming or panning occurs during animation, a `SET_IMAGE_VIEW` message is sent to the backend, updating the view bounds. These new bounds are used in the next frame generated by the backend.

2.3.4 Changing view parameters

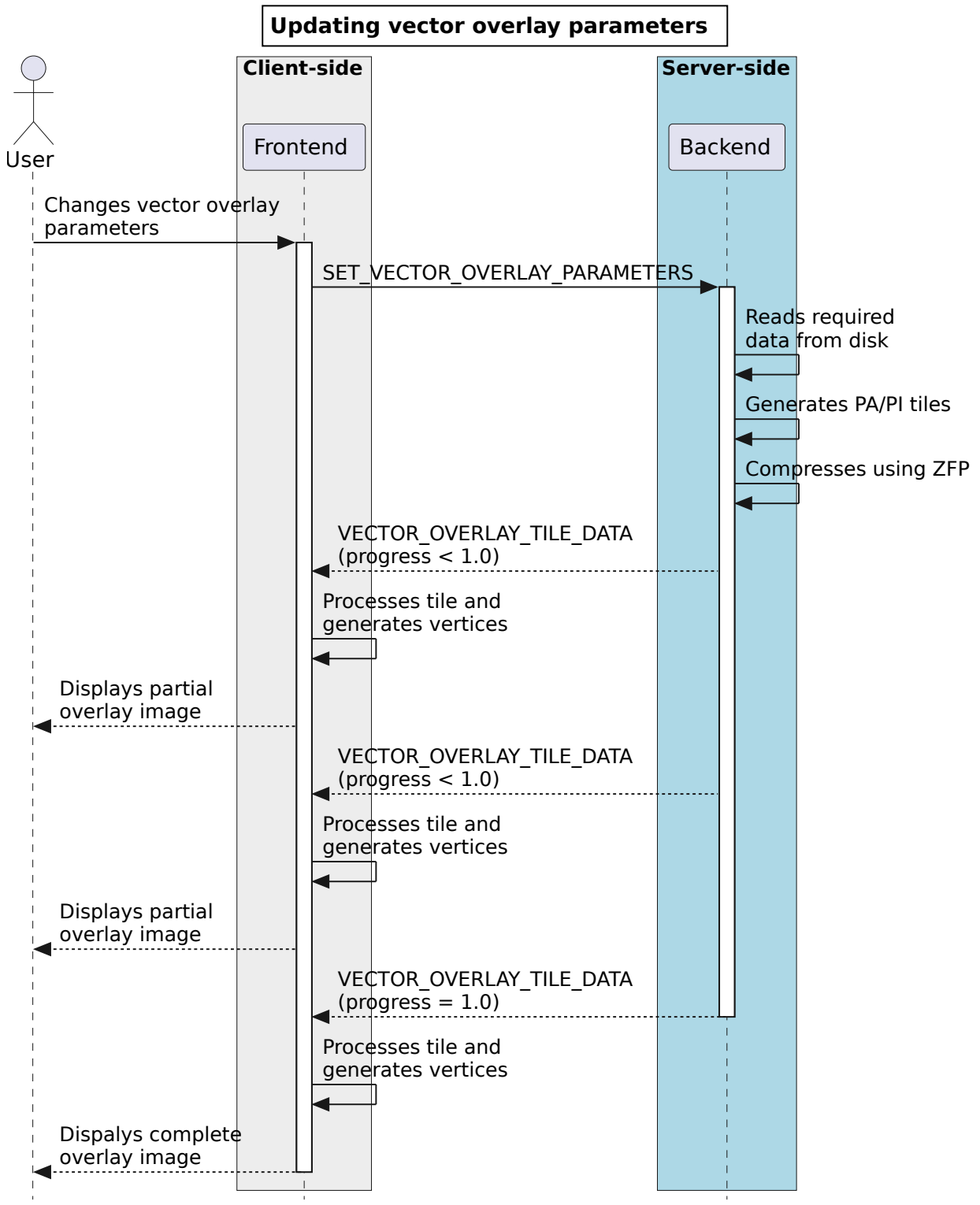
Contours must be re-calculated by the server when the contour parameters (levels, mode or smoothness) change. However, as contour rendering is done on the frontend, any changes to the contour rendering parameters (visibility, opacity, thickness, colour, line style) do not require any server interaction.



Similarly for raster images: As all the rendering is done on the frontend, any changes to the raster rendering configuration (colour map, range, scaling type) do not require any interaction between frontend and backend:



Vector overlay rendering requires image data for both the vector angle (normally calculated from polarization angle *PA*) and length/intensity (normally calculated from polarized intensity *PI*). The image data is first downsampled on the backend using block downsampling with an even block width, and then masked with a threshold value. Adjusting the block width or threshold value will require the data to be recalculated and streamed by the backend. The backend streams data tile-by-tile.

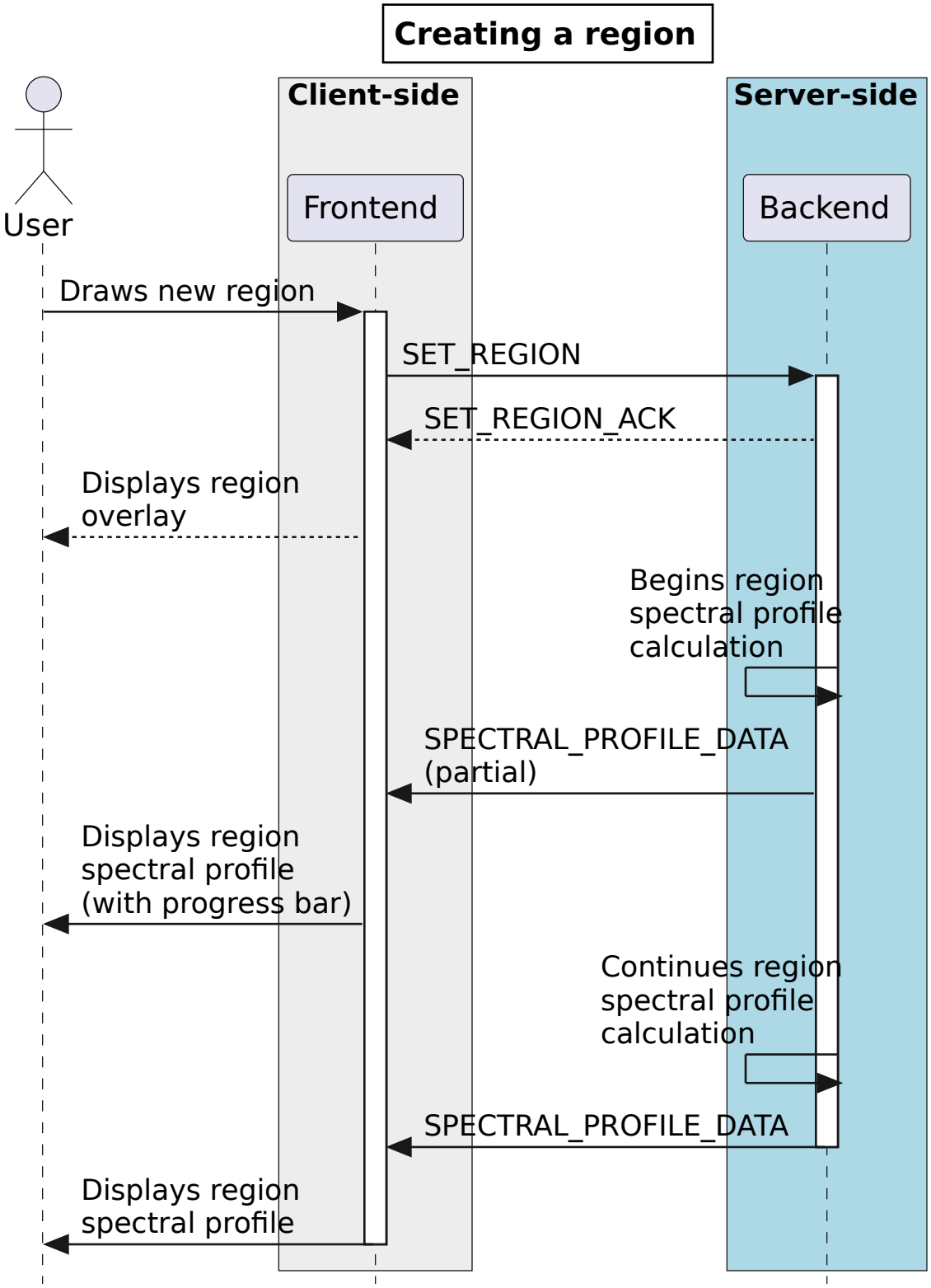

Updating vector overlay parameters
Client-side
Server-side
User
Frontend
Backend
Changes vector overlay parameters
SET_VECTOR_OVERLAY_PARAMETERS
Reads required data from disk
Generates PA/PI tiles
Compresses using ZFP
VECTOR_OVERLAY_TILE_DATA (progress < 1.0)
Processes tile and generates vertices
Displays partial overlay image
VECTOR_OVERLAY_TILE_DATA (progress < 1.0)
Processes tile and generates vertices
Displays partial overlay image
VECTOR_OVERLAY_TILE_DATA (progress = 1.0)
Processes tile and generates vertices
Dispalys complete overlay image

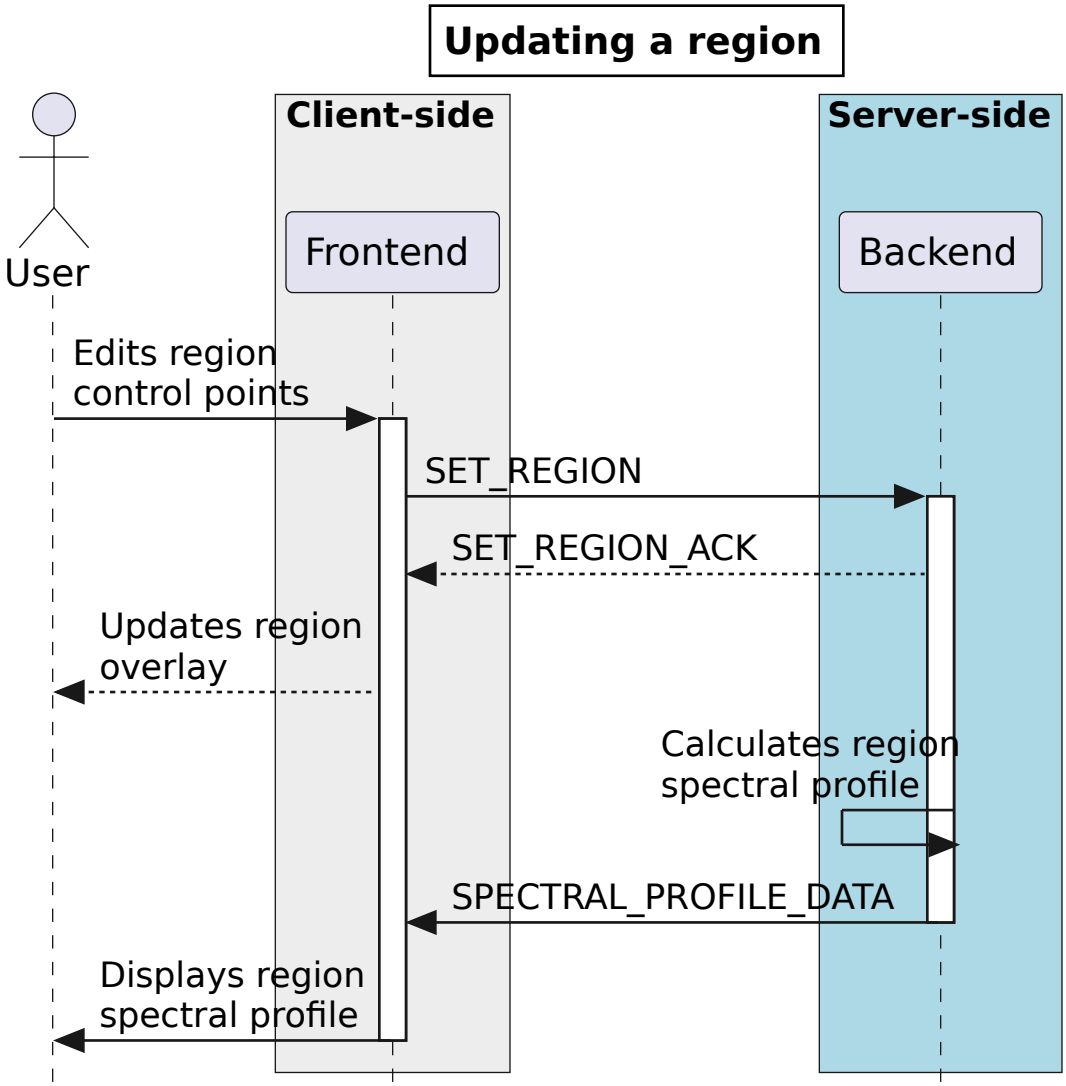
2.3.5 Region selection and statistics

Region creation

Regions can be created, removed and updated. Any profiles or statistics data associated with a region flow from the backend to the server whenever an update is required. Updates may be required (a) when a region is created or updated; (b) when the image channel is explicitly switched to a different channel or Stokes parameter using *SET_IMAGE_CHANNELS* or (c) when an animation playback results in the image view being updated implicitly.

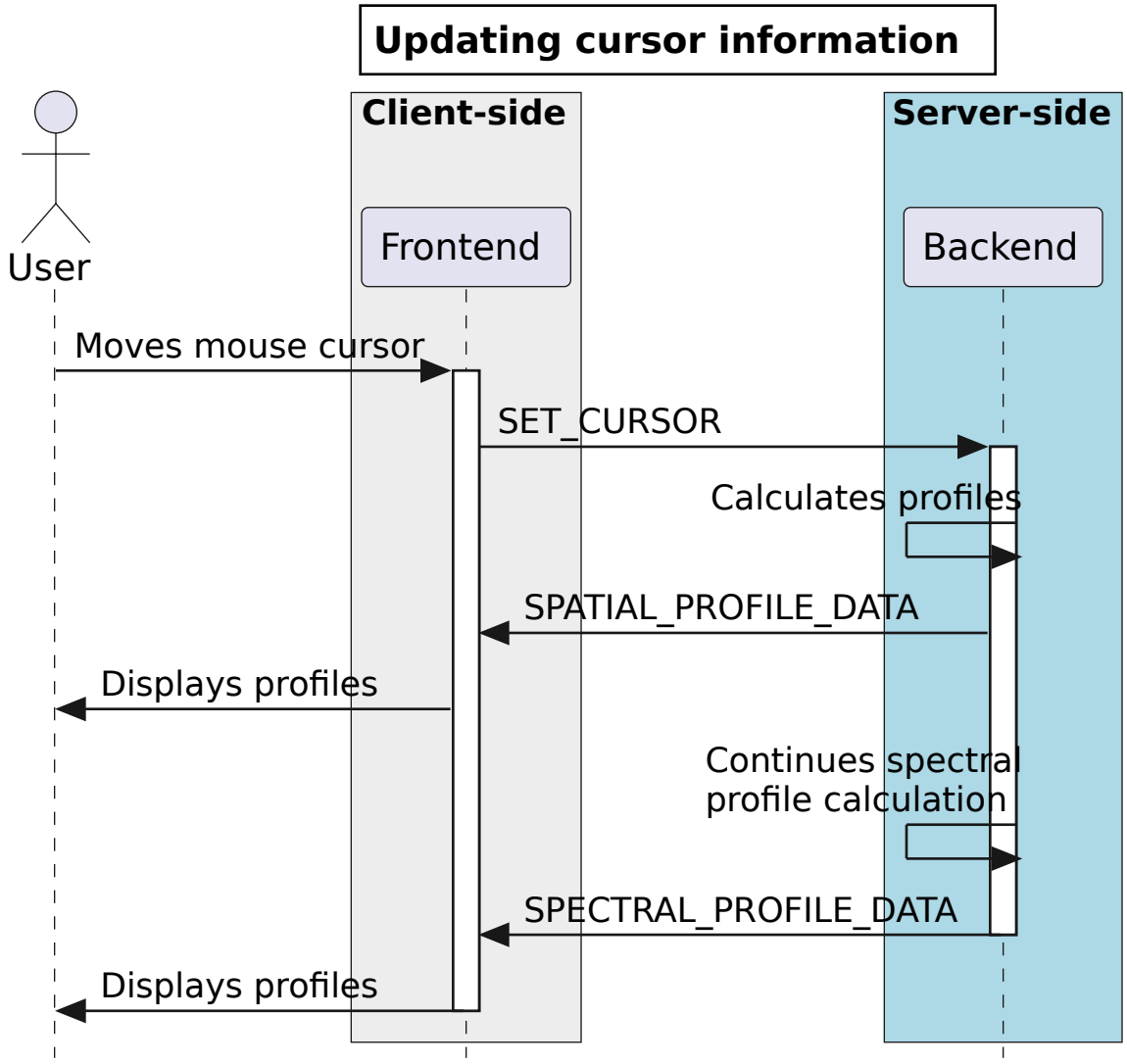
In addition, the backend may choose to provide partial region statistics or profile updates if the calculations are time-intensive. When creating a region, the `region_id` field of *SET_REGION* is less than zero: the backend generates the unique region_id field, and returns it in the acknowledgement message.


Creating a region
Client-side
Server-side
User
Frontend
Backend
Draws new region
SET_REGION
SET_REGION_ACK
Displays region overlay
Begins region spectral profile calculation
SPECTRAL_PROFILE_DATA (partial)
Displays region spectral profile (with progress bar)
Continues region spectral profile calculation
SPECTRAL_PROFILE_DATA
Displays region spectral profile



Cursor updates

As viewing profiles based on the position of the cursor is a very common use case, a separate control message is used specifically for this purpose, and does not require the definition of any additional region. The cursor-based region has a `region_id` field value of zero, and is defined as a point-type region. The X and Y coordinates of the region can only be updated via the *SET_CURSOR* command, while the channel and Stokes coordinates are automatically updated by the backend whenever the image view is changed.



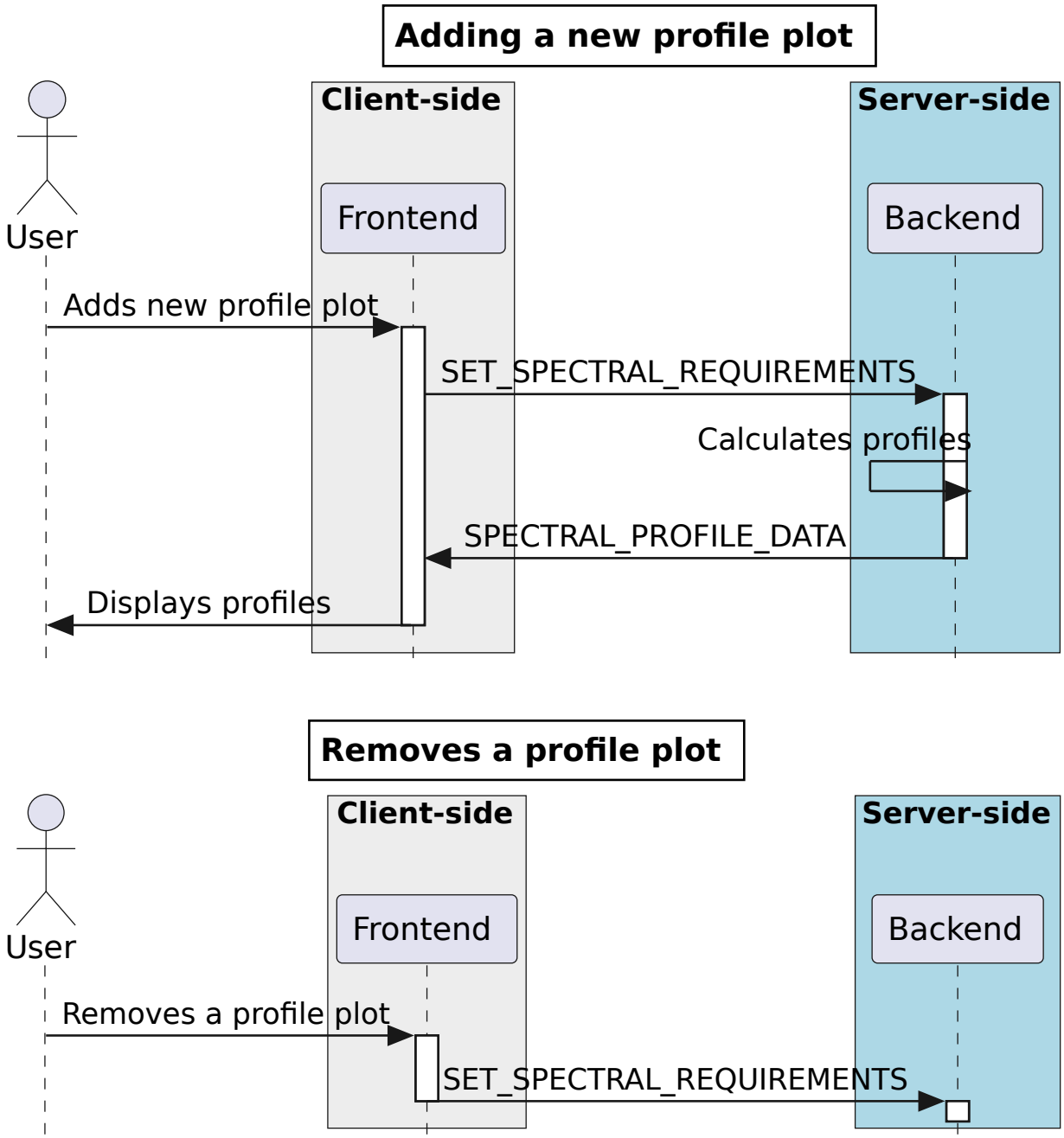
Region requirements

Each region can have analytical data requirements associated. For example, the user may wish to display the Z-profile of a particular region, while displaying the X- and Y-profiles of the cursor region. Whenever an analytical widget is added or removed in the frontend, the frontend must update the requirements associated with that region using the relevant command:

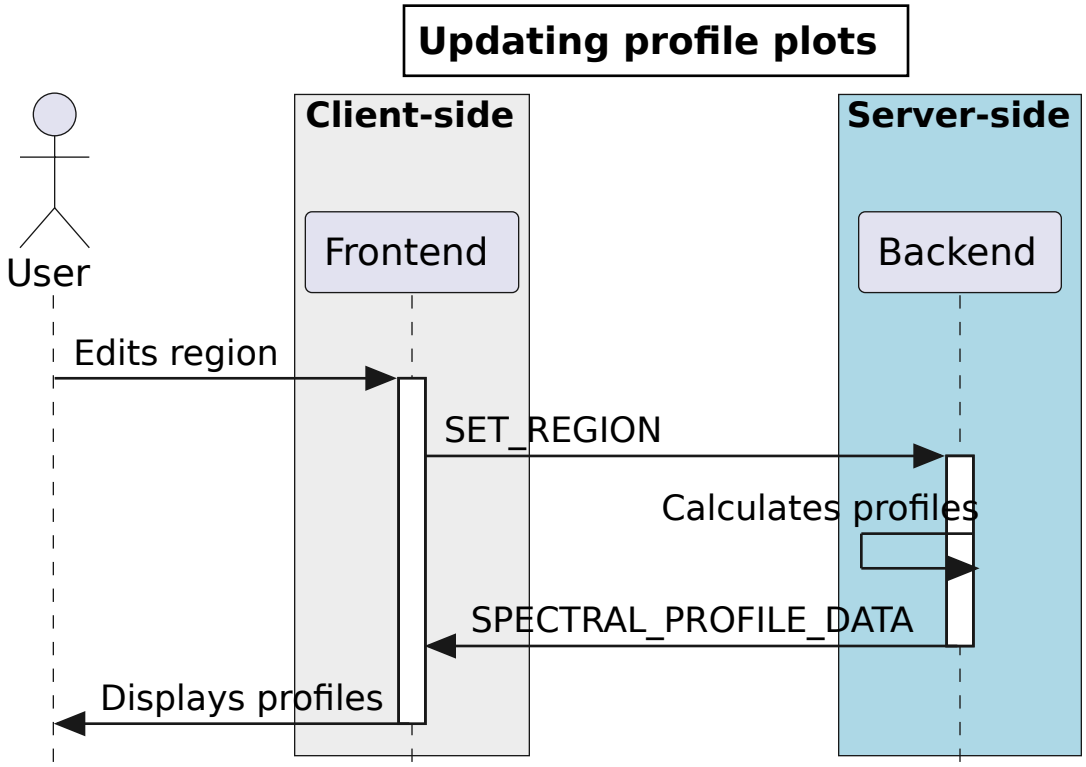
- `SET_SPECTRAL_REQUIREMENTS` for spectral profiler widgets
- `SET_SPATIAL_REQUIREMENTS` for spatial profiler widgets
- `SET_STATS_REQUIREMENTS` for stats info displays
- `SET_HISTOGRAM_REQUIREMENTS` for histograms plot widgets

After each requirements update, the backend should then assess the new requirements to determine whether any new or updated analytical data needs to be sent to the frontend. As an example: adding a spectral profile widget on the frontend and setting its requirements will mean that the region it is associated with now has an additional requirement, and the frontend requires new data. As such, the backend will calculate the required spectral profile and send it using

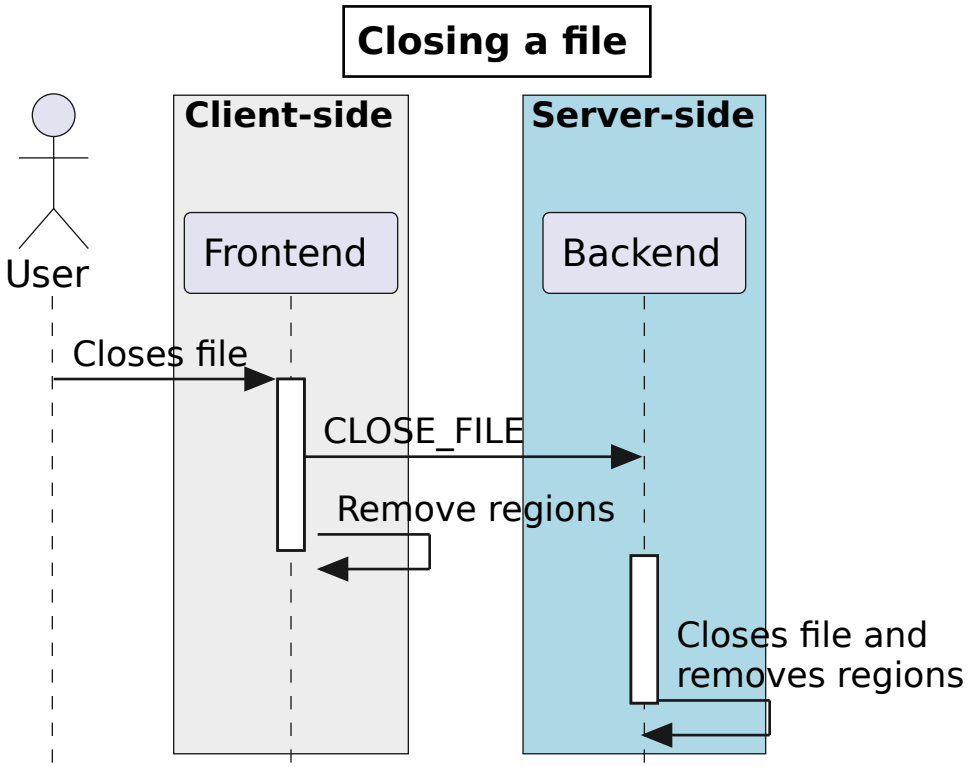
SPECTRAL_PROFILE_DATA. However, removing the spectral profile widget on the frontend will now remove that requirement, but no new *SPECTRAL_PROFILE_DATA* message is needed from the frontend.



If a region's parameters are changed, the backend determines which calculations need to be updated, based on the region's requirements set, and any required data is sent to the frontend through a new data stream message:



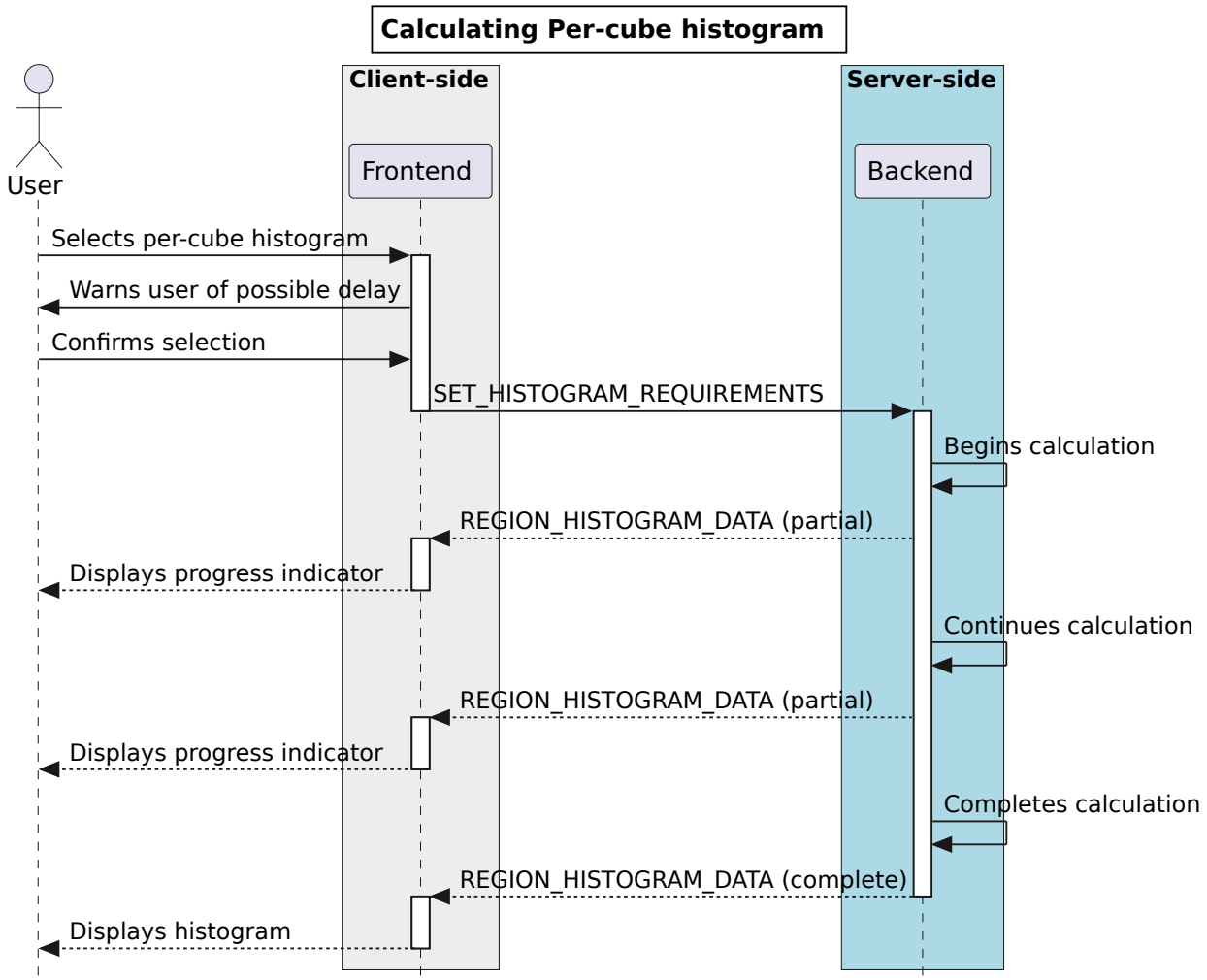
When all files are closed, regions associated with that file are removed, both on the frontend and on the backend. When only a single frame is closed, the regions persist.



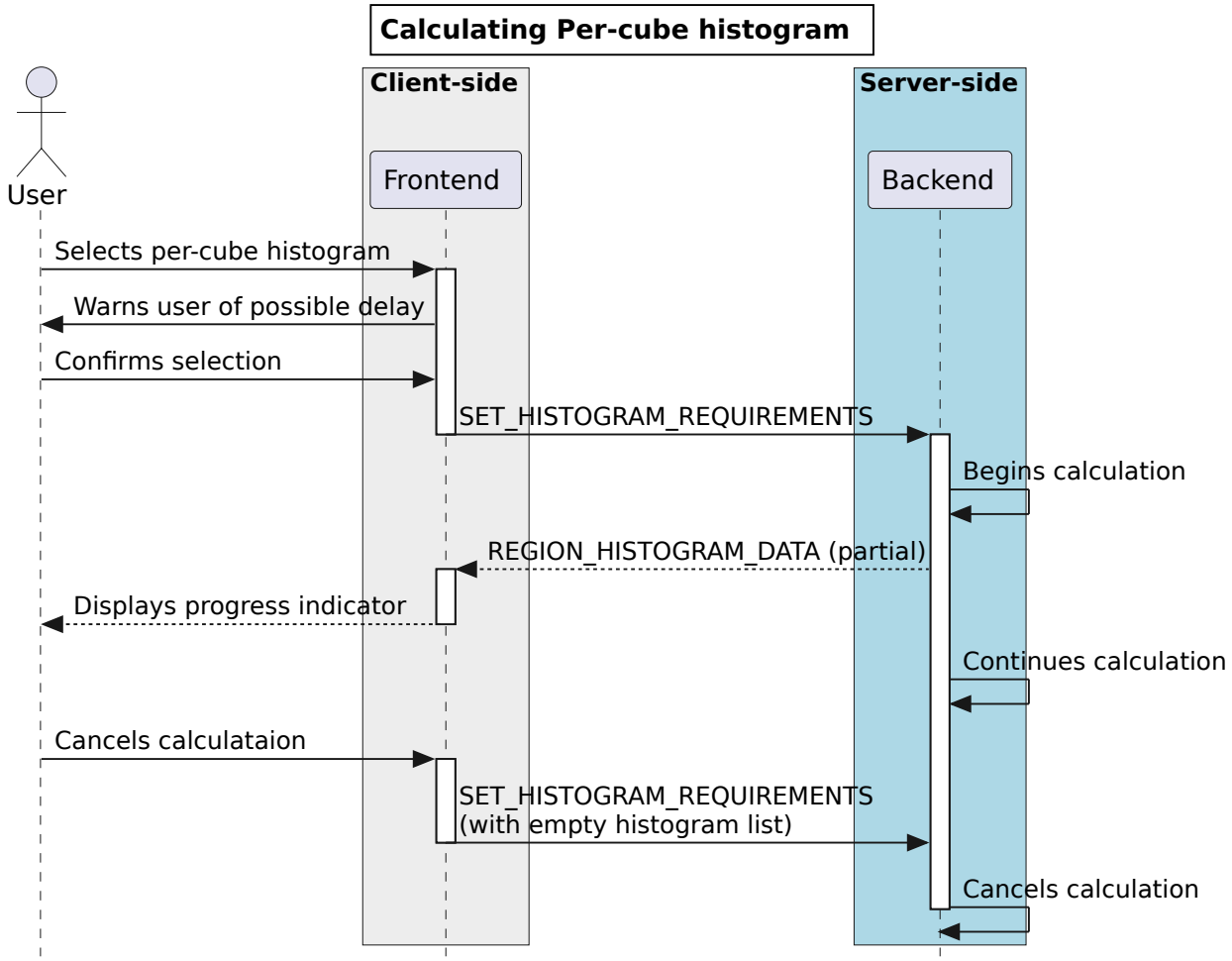
Per-cube histograms

As users may wish to use a histogram generated from the entire cube to choose their render bounds, the backend needs to support the calculation of a histogram on a per-cube as well as per-slice basis. A per-cube histogram is requested through the *SET_HISTOGRAM_REQUIREMENTS* message, with the region ID set to -2. As per-cube histograms may take a long time to calculate, there are additional requirements over and above per-slice histograms.

The backend should deliver results from the histogram calculation at regular intervals. As the histogram. As the histogram calculation consists of a large number of separable calculations (reading through individual slices to determine min/max, reading through individual slices to fill the histogram bins), the backend can split the calculation up into smaller tasks, and deliver cumulative results to the frontend.



The backend should be able to cancel the histogram calculation when receiving a specific message from the frontend. By sending a second *SET_HISTOGRAM_REQUIREMENTS* message to the backend, with the region ID set to -2 and an empty histogram list, the frontend can indicate to the backend that the per-cube histogram is no longer required, and the backend can cancel the calculation.



2.3.6 Data streaming

While some data flows can be described by a simple request/response approach, such as retrieving file lists or file information, other data flows require an asynchronous data stream approach. This need arises from situations where a single state change command corresponds to more than one response from the backend. For example, changing image channel would require each spatial profile associated with the active image channel to be updated, possibly resulting in more than one *SPATIAL_PROFILE_DATA* messages. Moving a region would require updating any analytics associated with the region. It is the backend's responsibility to correctly determine which analytic data needs to be updated whenever a control message is sent. It is essential that the backend only recalculates and sends data when needed. In order to do this, the backend must keep track of any updates to region requirements, and use these requirements to determine whether updates are needed. Region requirements will reflect the current frontend UI configuration. Changes to the frontend UI configuration (such as changing between "average" and "max" on a spectral profile widget) will result in new region requirements being sent to the backend, which will then be processed, resulting in new data being sent to the frontend when required.

Some examples of possible resultant data streams for control messages are given below:

- *SET_IMAGE_CHANNELS*: Changing either the channel or the Stokes parameter would require new image data to be sent, for both raster and contour images. Changing from one channel to another in the same Stokes cube could result in histograms, spatial profiles or region stats to require updating. Changing to a new stokes cube could also require spectral profiles to be updated. These updates will depend on the defined regions and defined region requirements.

- *START_ANIMATION*: Starting an animation will require new image data for each frame. In addition, since the animation playback may be across file, Stokes or channel parameters, the same data streams as those arising from *SET_IMAGE_CHANNELS* can occur.
- *SET_CURSOR*: Updating the cursor position is a special case of updating a region. As the cursor position is a point region, only spectral data and spatial data can require an update.
- *SET_REGION*: Creating a region will not result in any data streams, as the region's requirements will be empty by default. However, updating a regions parameters (other than region name) could result in spatial profiles (for open regions), spectral profiles, region stats and histograms (for closed and point regions) to be updated.
- *SET_STATS_REQUIREMENTS*: Updating stats requirements for a region can result in region stats data being updated.
- *SET_HISTOGRAM_REQUIREMENTS*: Updating histogram requirements for a region (either by updating the channel required for the histogram or by changing the histogram bin number) can result in histogram data being updated.
- *SET_SPATIAL_REQUIREMENTS*: Updating spatial profile requirements for a region can result in spatial profile data being updated.
- *SET_SPECTRAL_REQUIREMENTS*: Updating spectral profile requirements for a region (either by changing the coordinate required, such as "Qz" or "Uz", or by changing the statistic type used to generate the profile) can result in spectral profile data being updated.
- *SET_CONTOUR_PARAMETERS*: Updating contour parameters for a file will result in new contour image data being required.

2.3.7 User preferences

If the backend supports the *USER_PREFERENCES* server feature flag, the frontend will expect all the user's preferences (default settings, color maps, interaction preferences and others) to be included in the *REGISTER_VIEWER_ACK* message. Changes to the user preferences can be made by the frontend through the `SET_USER_PREFERENCES` control message. Each preference to be updated, along with the updated value, is stored as a map. User preference entries can be removed from the server by sending a `SET_USER_PREFERENCES` message with a map of preference keys with empty values.

If the backend supports the *USER_LAYOUTS* server feature flag, the frontend will expect all the user's custom UI layouts to be included in the *REGISTER_VIEWER_ACK* message. Changes to individual layouts (adding, updating or removing) are updated through the `SET_USER_LAYOUT` control message.

2.3.8 Resume the session

The basic idea is that, when the frontend reconnects to the backend (with *REGISTER_VIEWER*), it would also send some state information, such as:

- list of open files, along with their IDs and the current channels and stokes
- list of regions for each file, along with all their properties

Users can choose whether to resume the session while reconnected. If yes, then the backend would then reconstruct the session based on the frontend's message, by opening files again, changing to the appropriate channels, and so on, and then adding the regions and then set requirements.

There are two use cases for resuming with an existing session ID, and a third where resume is not possible.

1. Backend is restarted, frontend connects, frontend sends state information.
    1. Frontend sends *REGISTER_VIEWER* with session_id > 0.

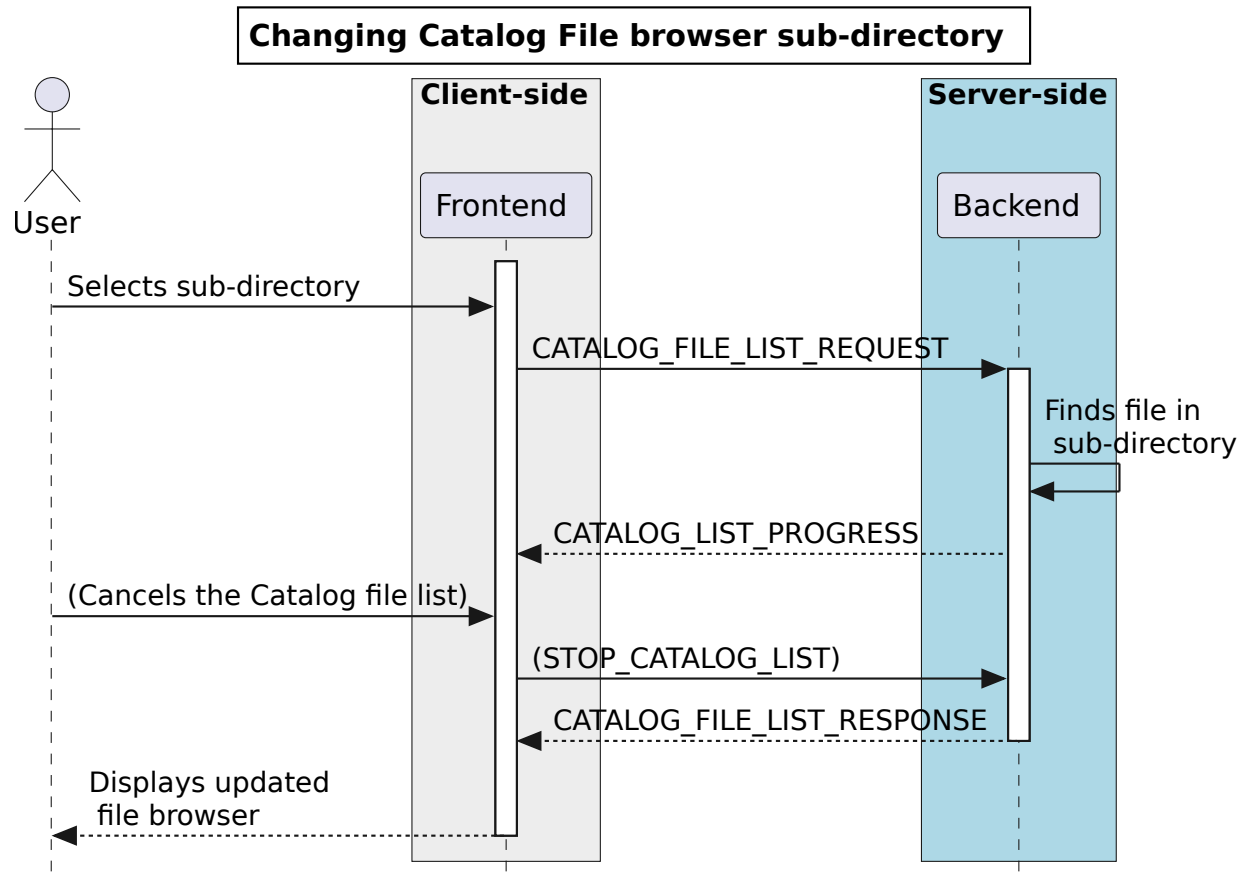
2. Restarted backend has no session_ids, *REGISTER_VIEWER_ACK* sets session_type=RESUMED. Backend creates new Session with given session_id (On Connect).
3. Frontend sends state to backend, i.e., sends *RESUME_SESSION* message with state information, backend responds with *RESUME_SESSION_ACK*.
4. Backend sets state in newly-created Session.

2. Network connection drops, frontend reconnects to backend with existing session id.
    1. While the network connection drops. It seems the uWebsocket has a default timeout setting for 15,000 ms (need to verify). For the new version of uWebsocket, we can set the timeout via the variable "*.idleTimeout"*. On Disconnect is called after the timeout and then backend deletes Session.
    2. Frontend sends *REGISTER_VIEWER* with session_id > 0.
    3. Backend has session_id, *REGISTER_VIEWER_ACK* sets session_type=RESUMED. Frontend sends state to backend with *RESUME_SESSION*, and backend responses with *RESUME_SESSION_ACK*.
    4. Backend sets state in existing Session, requirements trigger sending data streams (possibly cached).
3. Frontend is restarted, has no existing session id so cannot resume even though backend continues.
    1. Frontend sends *REGISTER_VIEWER* with session_id = 0.
    2. Backend creates a new Session, *REGISTER_VIEWER_ACK* sets session_type=NEW.
    3. The Session will be deleted immediately while the frontend is restarted.

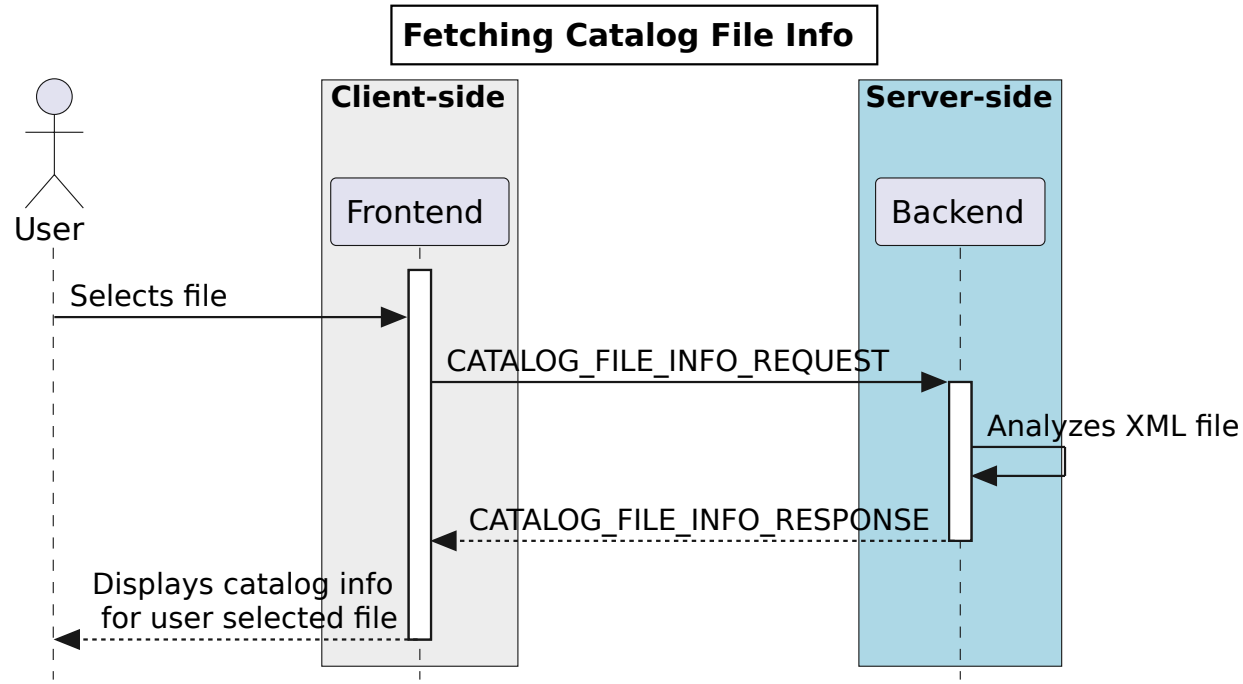
2.3.9 Catalog overlay

Sequence Diagrams

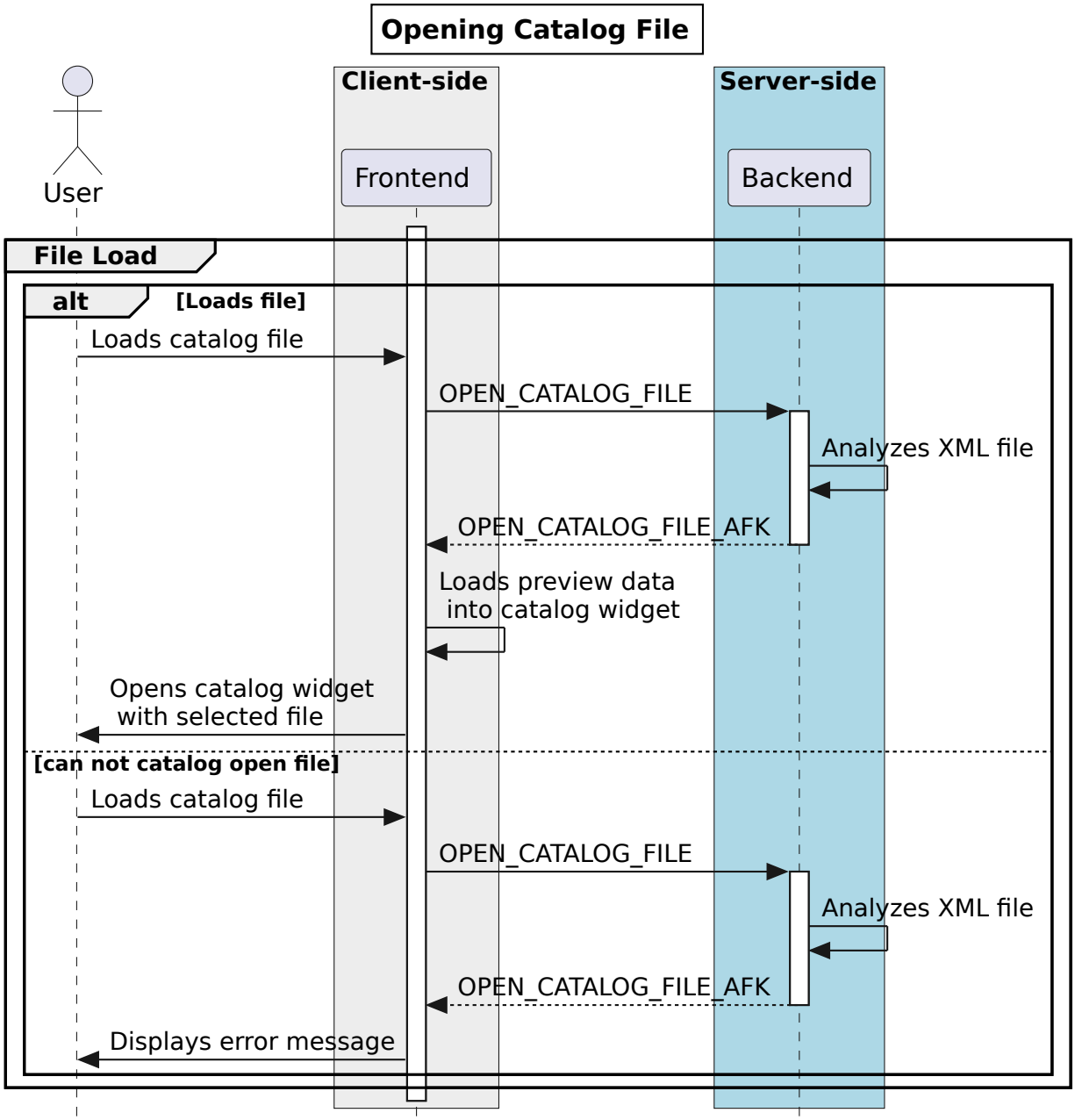
Catalog file list



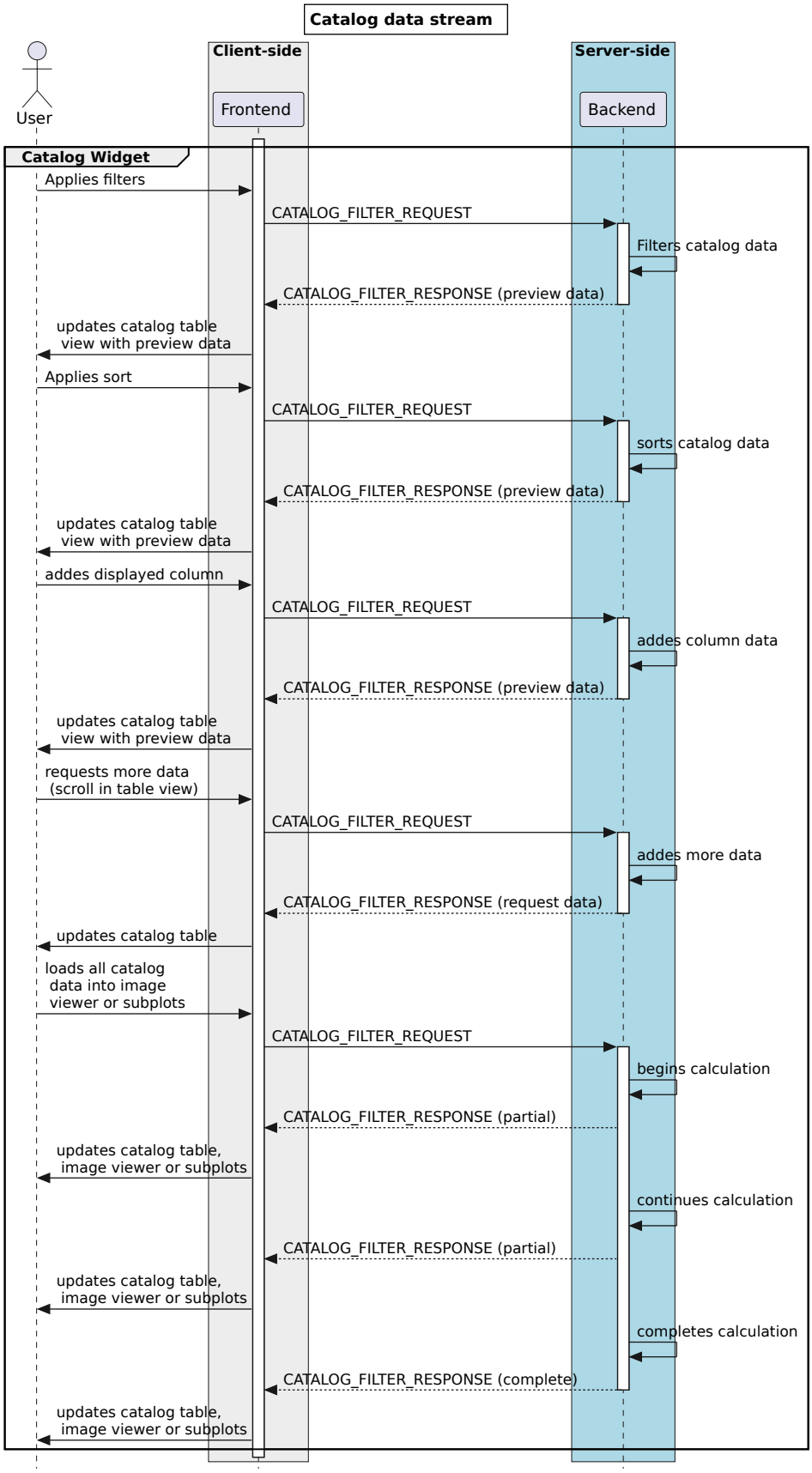
Catalog file info



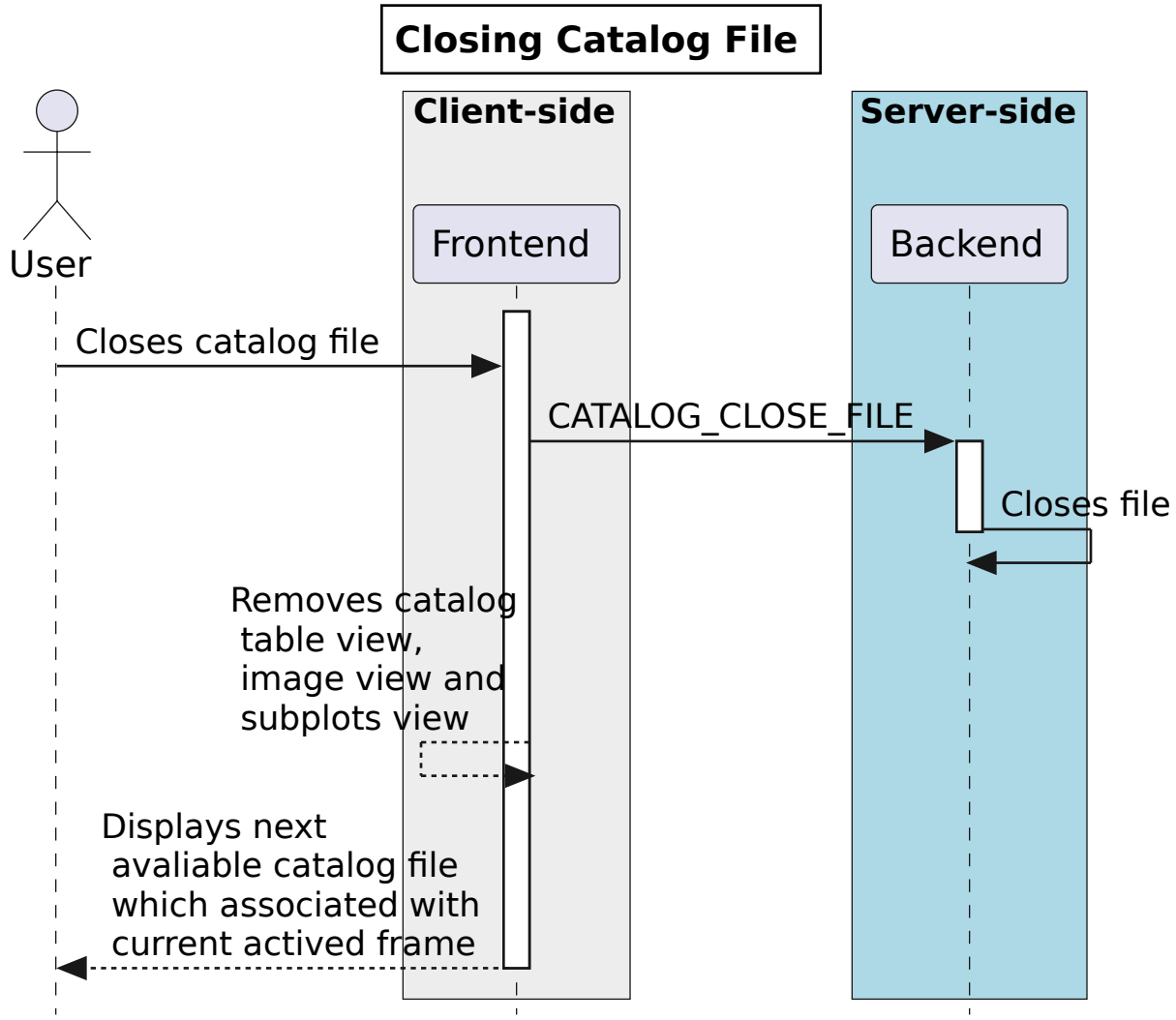
Opening catalog file



Catalog file data stream


Catalog data stream
Client-side
Server-side
User
Frontend
Backend
Catalog Widget
Applies filters
CATALOG_FILTER_REQUEST
Filters catalog data
CATALOG_FILTER_RESPONSE (preview data)
updates catalog table view with preview data
Applies sort
CATALOG_FILTER_REQUEST
sorts catalog data
CATALOG_FILTER_RESPONSE (preview data)
updates catalog table view with preview data
addes displayed column
CATALOG_FILTER_REQUEST
addes column data
CATALOG_FILTER_RESPONSE (preview data)
updates catalog table view with preview data
requests more data (scroll in table view)
CATALOG_FILTER_REQUEST
addes more data
CATALOG_FILTER_RESPONSE (request data)
updates catalog table
loads all catalog data into image viewer or subplots
CATALOG_FILTER_REQUEST
begins calculation
CATALOG_FILTER_RESPONSE (partial)
updates catalog table, image viewer or subplots
continues calculation
CATALOG_FILTER_RESPONSE (partial)
updates catalog table, image viewer or subplots
completes calculation
CATALOG_FILTER_RESPONSE (complete)
updates catalog table, image viewer or subplots

Closing catalog file



2.3.10 Moments generator

The moment generator should allow users to generate moment images from a cube interactively with the GUI. The interactivity should happen with a spectral line profile plot as usually we need information from spectral line profiles (line spectral/intensity distributions) to decide the control parameters of the moment generator. This could happen with the existing spectral profile widget, or, with a dedicated moment generator widget/dialogue with a spectral line profile plot.

CARTA should provide the following kinds of moments (sensible name in bold) as supported by CASA:

- moments = -1 - **mean value of the spectrum**
- moments = 0 - **integrated value of the spectrum**
- moments = 1 - **intensity weighted coordinate**; traditionally used to get "velocity fields"
- Moments = 2 - **intensity weighted dispersion of the coordinate**; traditionally used to get "velocity dispersion"
- moments = 3 - **median value of the spectrum**

- moments = 4 - **median coordinate**
- moments = 5 - **standard deviation about the mean of the spectrum**
- moments = 6 - **root mean square of the spectrum**
- moments = 7 - **absolute mean deviation of the spectrum**
- moments = 8 - **maximum value of the spectrum**
- moments = 9 - **coordinate of the maximum value of the spectrum**
- moments = 10 - **minimum value of the spectrum**
- moments = 11 - **coordinate of the minimum value of the spectrum**

The newly generated moment images (multiple moments can be generated at the same time) should be loaded and appended (and match spatially) in CARTA. CARTA should also support the capability to export the images as files in the following formats:

- CASA image format
- FITS image format
- HDF5-IDIA schema image format (TBD; post v1.4)

We create temporary moment images in the backend. Then if users want to keep the results, the "save image" option in the file menu should be used where filename and file type can be defined. If users don't do the "save image" step, those images should be deleted when the session is closed.

The interactivity with the spectral profile widget should include the following:

1. Text fields to specify spectral ranges to generate moments. This includes:

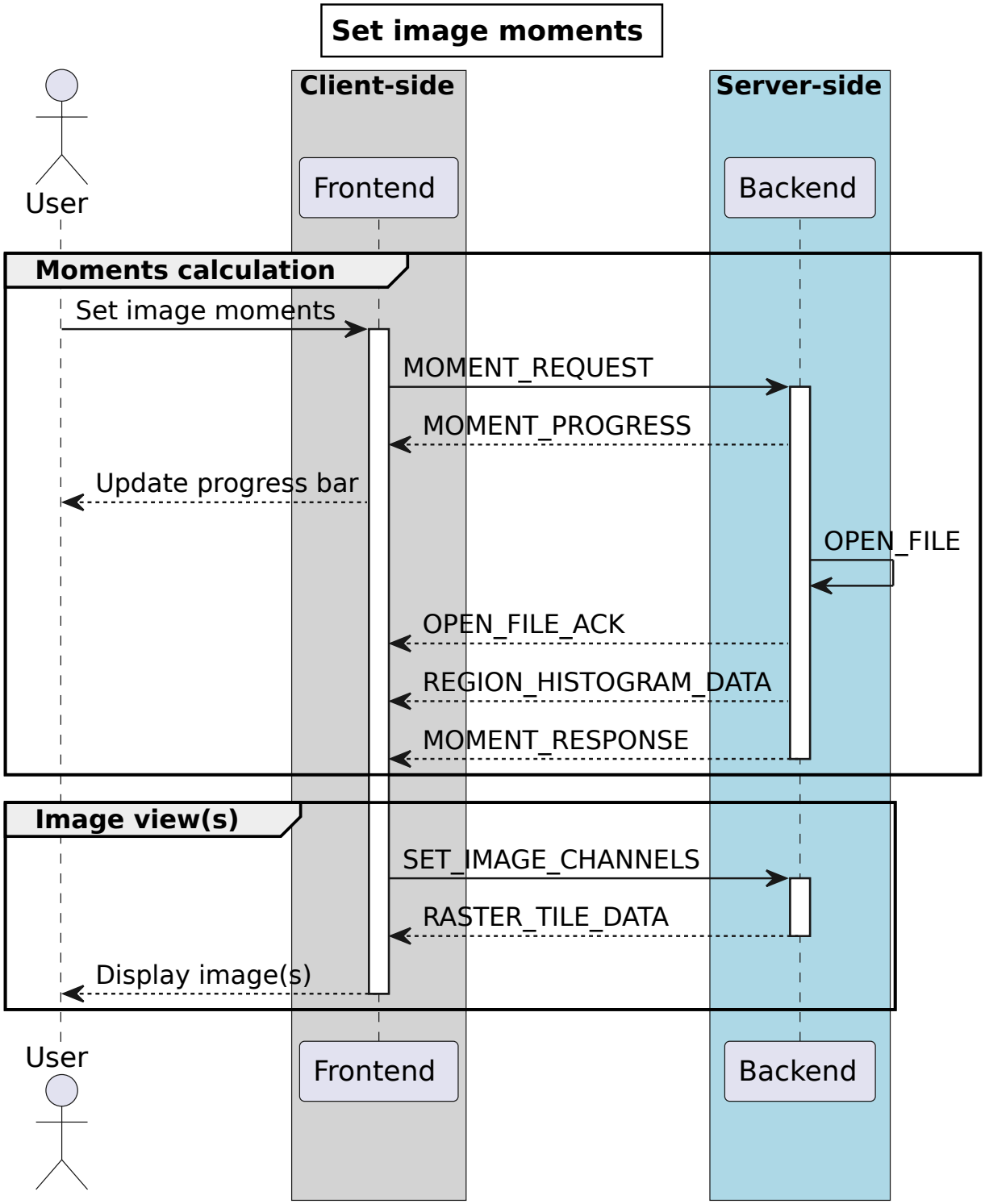
- Channel
- Velocity
- Frequency
- Stokes

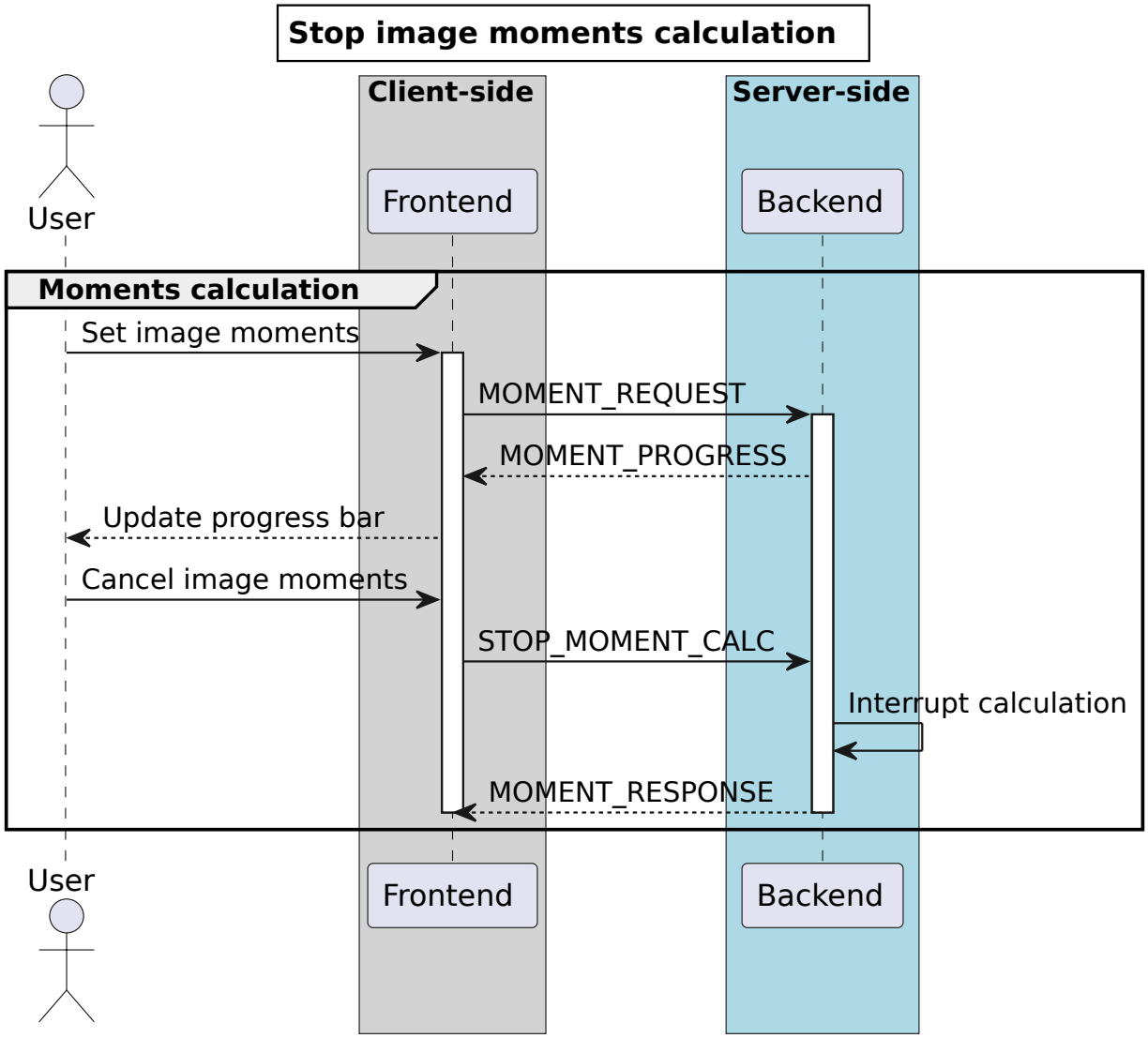
These text fields (except Stokes) are linked to the selection via the cursor directly on the spectral plot. Users can drag on the spectral plot to define a range in the spectral axis.

2. Text fields to define masks for the intensity values. Users can define a range of intensity values to be included in the moment calculations. For example, usually we will apply a threshold (e.g., >= 5-sigma) to the cube to compute moment 1 and moment 2. These text fields are linked to the selection via the cursor directly on the spectral plot. Users can drag on the spectral plot to define thresholds for moments.

As image cubes might be extremely large, the moment generator in CARTA should support an accurate progress bar (CASA provides "multiple" 0-100% progress bars which is misleading and does not provide useful information) and most importantly, the ability of cancellation.

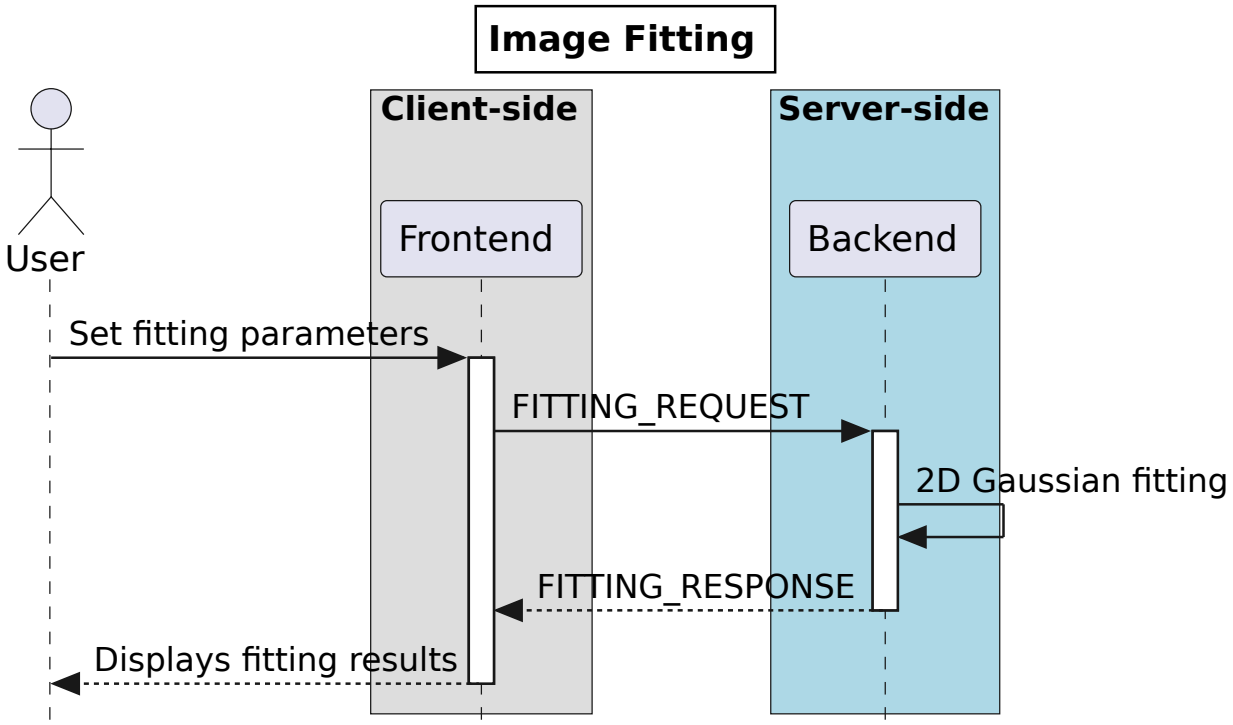
Sequence diagrams for setting image moments and stopping moments calculation are shown below:


Set image moments
Client-side
Server-side
User
Frontend
Backend
Moments calculation
Set image moments
MOMENT_REQUEST
MOMENT_PROGRESS
Update progress bar
OPEN_FILE
OPEN_FILE_ACK
REGION_HISTOGRAM_DATA
MOMENT_RESPONSE
Image view(s)
SET_IMAGE_CHANNELS
RASTER_TILE_DATA
Display image(s)
User
Frontend
Backend



2.3.11 Image fitting

Users can fit multiple 2D Gaussian components to the selected file with the image fitting widget. Frontend sends *FITTING_REQUEST* with `file_id` and initial values. Backend fits the current channel and polarization of the file and responds with *FITTING_RESPONSE*. The sequence diagram is shown below:



2.4 Layer descriptions

2.4.1 Application Layer

Interface communication messages fall into three overall categories:

- **Control messages** (along with any associated acknowledgement responses), which are used to modify the state of the backend from the frontend. Example of this would be starting a new session, moving the cursor or updating region parameters. Each message from the frontend correspond to zero or one acknowledgement response from the backend. Message names for this category follow the naming convention MESSAGE_NAME and MESSAGE_NAME_ACK
- **Request messages** (along with the required responses), which are used to explicitly request information from the backend without explicitly changing the backend state. Examples of this would be requesting a file list. The frontend will wait for a response for each request of this type, and callbacks or promises will be used to execute code based on the returned response. As each request needs to be mapped to response, messages in this category must include a unique requestID entry. Each message from the frontend in this category corresponds to exactly one response from the backend. Message names for this category follow the naming convention MESSAGE_NAME_REQUEST and MESSAGE_NAME_RESPONSE
- **Data flow messages**, which flow from the backend to the frontend without an originating front end request. These messages are used for pushing updated data from the backend to the frontend. Examples of this type would be image data, region statistics, profile data and cursor values. The appropriate mechanism for dealing with these messages in the frontend is a observable/subscription-based approach. As there is no request/response combination for messages in this category, there is no prescribed message naming convention.

Implementation note: The backend should implement a command queue for control messages, so that high priority messages are executed first, and cause the backend to disregard any queued-up control messages that are no longer relevant. As an example: moving the cursor across the image will result in a large number of control messages being

sent to the backend. Each of these control messages could result in a data flow message with new cursor and profile information, which may take some time to calculate. If a file is closed by the frontend, the backend no longer needs to process any remaining cursor messages relating to this file, and those messages should be removed from the queue.

Message definitions shown in blue are used for frontend ->backend communication. Message definitions shown in red are used for backend->frontend communication.

2.4.2 Presentation layer

Messages are encoded using the Protocol Buffers message format, which encodes into a binary format. Each message is prepended by a 64-bit structure, consisting of:

- 16-bit unsigned integer, used to identify the message type, specified by *EventType*
- 16-bit unsigned integer, used to determine the ICD version
- 32-bit unsigned integer, used to uniquely identify requests and corresponding responses. In the case of messages with no corresponding request, such as data stream messages, this integer will be ignored.

Using an 8-byte header prevents byte alignment issues from cropping up. End points decode the message by splitting it into two sections: the 8-byte identifier header and the payload. The identifier header is used to determine which Protocol Buffer definition should be used to decode the payload, and which request corresponds to which response. The ICD version integer (shown at the top of this document) should match the major version of this document (also shown at the top of this document). Any changes to the protocol buffer definitions that would render older backend or frontend implementations incompatible should result in incrementing the ICD version number, and a corresponding change to this document's version number.

Implementation note: The protocol buffer style guide [6] expects snake_case for field names. The protobuf c++ compiler leaves names in snake_case, while the javascript compiler leaves field names in camelCase. So a field accessed via `msg.min_val() in c++` would be accessed by `msg.minVal` in javascript.

2.4.3 Session Layer

Sessions will utilise the the WebSocket protocol, as the frontend will be browser-based. Initial session establishment will occur using HTTP, and then be upgraded to WebSocket. Session management will be handled by a session ID being passed from backend to frontend on initial connection. If the frontend is disconnected without closing the session explicitly, the session ID can be passed to the backend upon reconnection to resume the session, although this is not currently supported.

2.4.4 Transport Layer

The interface will use TCP to communicate. Network layer and below will be dependent on the server/client connection and need not be detailed.

2.5 Protocol buffer reference

2.5.1 Messages

AddRequiredTiles

Source file: control/tiles.proto

ADD_REQUIRED_TILES Provides a list of tiles that are required for the specified file

Field	Type	Label	Description
file_id	sfixed32		The file ID that the view corresponds to
tiles	sfixed32	repeated	The list of tiles required, in encoded coordinate
compression_type	*CompressionType*		The compression algorithm used
compression_quality	float		Compression quality switch

AnimationFlowControl

Source file: control/animation.proto

ANIMATION_FLOW_CONTROL Used for informing the backend of which frames have been received

Field	Type	Label	Description
file_id	sfixed32		
received_frame	*AnimationFrame*		The latest flow control frame received
animation_id	sfixed32		The animation ID that the flow control message belongs to
timestamp	sfixed64		Timestamp at which the frame was received

CatalogFileInfoRequest

Source file: request/catalog_file_info.proto

Field	Type	Label	Description
directory	string		
name	string		

CatalogFileInfoResponse

Source file: request/catalog_file_info.proto

Field	Type	Label	Description
success	bool		
message	string		
file_info	*CatalogFileInfo*		
headers	*CatalogHeader*	repeated	

CatalogFilterRequest

Source file: stream/catalog_filter.proto

Field	Type	Label	Description
file_id	sfixed32		
column_indices	int32	repeated	
filter_configs	*FilterConfig*	repeated	
subset_data_size	sfixed32		
subset_start_index	sfixed32		
image_bounds	*CatalogImageBounds*		
image_file_id	sfixed32		
region_id	sfixed32		
sort_column	string		
sorting_type	*SortingType*		

CatalogFilterResponse

Source file: stream/catalog_filter.proto

Field	Type	Label	Description
file_id	sfixed32		
image_file_id	sfixed32		
region_id	sfixed32		
columns	map<key: fixed32, value: *ColumnData*>	repeated	
subset_data_size	sfixed32		
subset_end_index	sfixed32		
progress	float		
filter_data_size	sfixed32		
request_end_index	sfixed32		

CatalogListRequest

Source file: request/catalog_list.proto

Field	Type	Label	Description
directory	string		
filter_mode	*FileListFilterMode*		Filter mode to use when showing the file list

CatalogListResponse

Source file: request/catalog_list.proto

Field	Type	Label	Description
success	bool		
message	string		
directory	string		
parent	string		
files	*CatalogFileInfo*	repeated	
subdirectories	*DirectoryInfo*	repeated	
cancel	bool		

CloseCatalogFile

Source file: control/open_catalog_file.proto

Field	Type	Label	Description
file_id	sfixed32		

CloseFile

Source file: control/close_file.proto

CLOSE_FILE: Instructs the backend to close a file with a given file ID

Field	Type	Label	Description
file_id	sfixed32		Which "file" slot to close

ConcatStokesFiles

Source file: control/concat_stokes_files.proto

CONCAT_STOKES_FILES: Requests to concatenate individual stokes images as one and open it. Backend responds with *CONCAT_STOKES_FILES_ACK*

Field	Type	Label	Description
stokes_files	*StokesFile*	repeated	Stokes files to be concatenated
file_id	sfixed32		File ID for the concatenate image
render_mode	*RenderMode*		The render mode to use. Additional modes will be added in subsequent versions.

ConcatStokesFilesAck

Source file: control/concat_stokes_files.proto

Field	Type	Label	Description
success	bool		Concatenation is successful or not
message	string		Error message if not successful
open_file_ack	*OpenFileAck*		Open file acknowledgement for the concatenate file

ContourImageData

Source file: stream/contour_image.proto

CONTOUR_IMAGE_DATA: Data for an image rendered in contour mode.

Field	Type	Label	Description
file_id	sfixed32		The file ID that the contour image corresponds to
reference_file_id	fixed32		The file ID of the reference image that the contour vertices are mapped to
image_bounds	*ImageBounds*		The bounding box in the XY plane corresponding to the image data in pixel coordinates
channel	sfixed32		The image channel used to generate the contours
stokes	sfixed32		The image stokes parameter used to generate the contours
contour_sets	*ContourSet*	repeated	Each contour set consists of the contour level value, as well as a list of coordinates. The start_indices list is used to determine how to subdivide the coordinates list into separate poly-lines when rendering.
progress	double		Progress of the contour sets being sent. If this is zero, the message is assumed to contain the entire contour sets

ContourSet

Source file: stream/contour_image.proto

Field	Type	Label	Description
level	double		
decimation_factor	int32		
raw_coordinates	bytes		
raw_start_indices	bytes		
uncompressed_coordinates_size	int32		

ErrorData

Source file: stream/error.proto

ERROR_DATA: Stream of error/warning/info data. This stream is used to present the frontend with additional information on the state of the backend, and is not used in place of returning success=false on requests or commands.

Field	Type	Label	Description
severity	*ErrorSeverity*		The severity of the error. Critical errors are reserved for errors that would normally require the user to restart the program or reload the page
tags	string	repeated	A list of strings describing the error type, that the frontend can interpret and react to. For example, "file_io" or "memory".
message	string		The error message
data	string		Accompanying error data. For example, if an error has the "file_io" tag, the frontend would expect the data field to contain the file ID of the offending file.

ExportRegion

Source file: control/export_region.proto

EXPORT_REGION: Requests exporting the specified regions to a file on the server. If directory and file are blank, return file contents for export on client. Backend responds with *EXPORT_REGION_ACK*

Field	Type	Label	Description
type	*FileType*		Required file type
coord_type	*CoordinateType*		Required coordinate type pixel/world
file_id	sfixed32		File id for the coordinate system to use
region_styles	map<key: sfixed32, value: *RegionStyle*>	repeated	Region ids and style params to export
directory	string		Optional directory name of server file
file	string		Optional file name of server file

ExportRegionAck

Source file: control/export_region.proto

EXPORT_REGION_ACK Response for *EXPORT_REGION* to indicate success and file contents if on client.

Field	Type	Label	Description
success	bool		Defines whether *EXPORT_REGION* was successful
message	string		Error message (if applicable)
contents	string	repeated	File contents for client export (one line per string)

FileInfoRequest

Source file: request/file_info.proto

FILE_INFO_REQUEST: Requests the file info for a specific file. Backend responds with *FILE_INFO_RESPONSE*

Field	Type	Label	Description
directory	string		Required directory name
file	string		Required file name
hdu	string		Required HDU name (if applicable). If left empty, the first HDU is selected

FileInfoResponse

Source file: request/file_info.proto

FILE_INFO_RESPONSE Response for *FILE_INFO_REQUEST*. Gives information on the requested file

Field	Type	Label	Description
success	bool		Defines whether the *FILE_INFO_REQUEST* was successful
message	string		Error message (if applicable)
file_info	*FileInfo*		Basic file info (type, size)
file_info_extended	map<key: string, value: *FileInfoExtended*>	repeated	Extended file info (WCS, header info)

FileListRequest

Source file: request/file_list.proto

FILE_LIST_REQUEST: Requests the list of available files for a given directory. Backend responds with *FILE_LIST_RESPONSE*

Field	Type	Label	Description
directory	string		Required directory name
filter_mode	*FileListFilterMode*		Filter mode to use when showing the file list

FileListResponse

Source file: request/file_list.proto

FILE_LIST_RESPONSE Response for *FILE_LIST_REQUEST*. Gives a list of available files (and their types), as well as subdirectories

Field	Type	Label	Description
success	bool		Defines whether the *FILE_LIST_REQUEST* was successful
message	string		Error message (if applicable)
directory	string		Directory of listing
parent	string		Directory parent (null/empty if top-level)
files	*FileInfo*	repeated	List of available image files, with file type information and size information.
subdirectories	*DirectoryInfo*	repeated	List of available subdirectories, with number of items and modified date
cancel	bool		

FittingRequest

Source file: request/fitting_request.proto

FITTING_REQUEST: Requests 2D Gaussian image fitting with given initial values. Backend responds with *FITTING_RESPONSE*

Field	Type	Label	Description
file_id	sfixed32		File ID of the image to be fit
initial_values	*GaussianComponent*	repeated	Initial values for 2D Gaussian fitting
fixed_params	bool	repeated	Whether each parameter (in the order of center, amplitude, FWHM, and p.a.) should be fixed when fitting

FittingResponse

Source file: request/fitting_request.proto

FITTING_RESPONSE: Response for *FITTING_REQUEST*. Gives results and log of 2D Gaussian image fitting.

Field	Type	Label	Description
success	bool		Defines whether *FITTING_REQUEST* was successful
message	string		Error message (if applicable)
result_values	*GaussianComponent*	repeated	Fitting result: values of the fitted parameters
result_errors	*GaussianComponent*	repeated	Fitting result: errors of the fitted parameters
log	string		Fitting log

HistogramConfig

Source file: control/region_requirements.proto

Field	Type	Label	Description
coordinate	string		
channel	sfixed32		
num_bins	sfixed32		

ImageProperties

Source file: control/resume_session.proto

Field	Type	Label	Description
directory	string		
file	string		
hdu	string		
file_id	sfixed32		
render_mode	*RenderMode*		
channel	sfixed32		
stokes	sfixed32		
regions	map<key: sfixed32, value: *RegionInfo*>	repeated	
contour_settings	*SetContourParameters*		
stokes_files	*StokesFile*	repeated	

ImportRegion

Source file: control/import_region.proto

IMPORT_REGION: Requests the opening and applying of a specific region file. Backend responds with *IMPORT_REGION_ACK*

Field	Type	Label	Description
group_id	sfixed32		Required WCS group id (may be a single file id)
type	*FileType*		Required file type
directory	string		Optional directory name of server file
file	string		Optional file name of server file
contents	string	repeated	Optional file contents of client file (1 line per string)

ImportRegionAck

Source file: control/import_region.proto

IMPORT_REGION_ACK Response for *IMPORT_REGION*. Also supplies region properties

Field	Type	Label	Description
success	bool		Defines whether *IMPORT_REGION* was successful
message	string		Error message (if applicable)
regions	map<key: sfixed32, value: *RegionInfo*>	repeated	Map region id to parameters
region_styles	map<key: sfixed32, value: *RegionStyle*>	repeated	Map region id to style parameters

MomentProgress

Source file: request/moment_request.proto

Field	Type	Label	Description
file_id	sfixed32		
progress	float		

MomentRequest

Source file: request/moment_request.proto

Field	Type	Label	Description
file_id	sfixed32		
moments	*Moment*	repeated	
axis	*MomentAxis*		
region_id	sfixed32		
spectral_range	*IntBounds*		
mask	*MomentMask*		
pixel_range	*FloatBounds*		

MomentResponse

Source file: request/moment_request.proto

Field	Type	Label	Description
success	bool		
message	string		
open_file_acks	*OpenFileAck*	repeated	
cancel	bool		

OpenCatalogFile

Source file: control/open_catalog_file.proto

Field	Type	Label	Description
directory	string		
name	string		
file_id	sfixed32		
preview_data_size	sfixed32		

OpenCatalogFileAck

Source file: control/open_catalog_file.proto

Field	Type	Label	Description
success	bool		
message	string		
file_id	sfixed32		
file_info	*CatalogFileInfo*		
data_size	sfixed32		
headers	*CatalogHeader*	repeated	
preview_data	map<key: fixed32, value: *ColumnData*>	repeated	

OpenFile

Source file: control/open_file.proto

OPEN_FILE: Requests the opening of a specific file. Backend responds with *OPEN_FILE_ACK*

Field	Type	Label	Description
directory	string		Required directory name
file	string		File name or LEL expression
hdu	string		Which HDU to load (if applicable). If left blank, the first HDU will be used
file_id	sfixed32		Which "file" slot to load the file into (when viewing multiple files)
render_mode	*RenderMode*		The render mode to use. Additional modes will be added in subsequent versions.
lel_expr	bool		Defines whether file is LEL expression

OpenFileAck

Source file: control/open_file.proto

OPEN_FILE_ACK Response for *OPEN_FILE*. Also supplies file information

Field	Type	Label	Description
success	bool		Defines whether *OPEN_FILE* was successful
file_id	sfixed32		Which file slot the file was loaded into (when viewing multiple files)
message	string		Error message (if applicable)
file_info	*FileInfo*		Basic file info (type, size)
file_info_extended	*FileInfoExtended*		Extended file info (WCS, header info)
file_feature_flags	fixed32		Optional bitflags specifying feature flags of the file being opened.
beam_table	*Beam*	repeated	*Beam* table for multiple-beam images

PvProgress

Source file: request/pv_request.proto

Field	Type	Label	Description
file_id	sfixed32		
progress	float		

PvRequest

Source file: request/pv_request.proto

Field	Type	Label	Description
file_id	sfixed32		
region_id	sfixed32		
width	sfixed32		

PvResponse

Source file: request/pv_request.proto

Field	Type	Label	Description
success	bool		
message	string		
open_file_ack	*OpenFileAck*		
cancel	bool		

RasterTileData

Source file: stream/raster_tile.proto

Field	Type	Label	Description
file_id	sfixed32		The file ID that the raster image corresponds to
channel	sfixed32		The image channel (z-coordinate)
stokes	sfixed32		The image stokes coordinate
compression_type	*CompressionType*		The compression algorithm used.
compression_quality	float		Compression quality switch
animation_id	sfixed32		The ID of the animation (if any)
tiles	*TileData*	repeated	List of tile data

RasterTileSync

Source file: stream/raster_tile.proto

Field	Type	Label	Description
file_id	sfixed32		The file ID that the raster image corresponds to
channel	sfixed32		The image channel (z-coordinate)
stokes	sfixed32		The image stokes coordinate
animation_id	sfixed32		The ID of the animation (if any)
end_sync	bool		Is this a start or end sync message?

RegionFileInfoRequest

Source file: request/region_file_info.proto

REGION_FILE_INFO_REQUEST: Requests contents for a specific region file on the server Backend responds with *REGION_FILE_INFO_RESPONSE*

Field	Type	Label	Description
directory	string		Required directory name
file	string		Required file name

RegionFileInfoResponse

Source file: request/region_file_info.proto

REGION_FILE_INFO_RESPONSE Response for *REGION_FILE_INFO_REQUEST*. Gives information on the requested file

Field	Type	Label	Description
success	bool		Defines whether the REGION_INFO_REQUEST was successful
message	string		Error message (if applicable)
file_info	*FileInfo*		Basic info about region file
contents	string	repeated	Contents of file; each string is a line

RegionHistogramData

Source file: stream/region_histogram.proto

REGION_HISTOGRAM_DATA: Stats data for a specific region

Field	Type	Label	Description
file_id	sfixed32		The file ID that the histogram corresponds to
region_id	sfixed32		The region ID corresponding to the histogram. If the histogram corresponds to the entire current 2D image, the region ID has a value of -1.
channel	sfixed32		The image channel corresponding to the histogram
stokes	sfixed32		The image stokes corresponding to the histogram
histograms	*Histogram*		Array of histograms of the current file, region, channel and stokes
progress	float		Progress indicator, in the case of partial histogram results being sent

RegionListRequest

Source file: request/region_list.proto

REGION_LIST_REQUEST: Requests the list of available region files for a given directory. Backend responds with *REGION_LIST_RESPONSE*

Field	Type	Label	Description
directory	string		Required directory name
filter_mode	*FileListFilterMode*		Filter mode to use when showing the file list

RegionListResponse

Source file: request/region_list.proto

REGION_LIST_RESPONSE Response for *REGION_LIST_REQUEST*. Gives a list of available region files (and their types), as well as subdirectories

Field	Type	Label	Description
success	bool		Defines whether the *REGION_LIST_REQUEST* was successful
message	string		Error message (if applicable)
directory	string		Directory of listing
parent	string		Directory parent (null/empty if top-level)
files	*FileInfo*	repeated	List of available image files, with file type information and size information.
subdirectories	*DirectoryInfo*	repeated	List of available subdirectories, with number of items and modified date
cancel	bool		

RegionStatsData

Source file: stream/region_stats.proto

REGION_STATS_DATA: Stats data for a specific region

Field	Type	Label	Description
file_id	sfixed32		The file ID that the profile corresponds to
region_id	sfixed32		The region_id corresponding to this profile. If the statistics data corresponds to the entire current 2D image, the region ID has a value of -1.
channel	sfixed32		The image channel used to generate the statistics
stokes	sfixed32		The image stokes parameter used to generate the profiles
statistics	*StatisticsValue*	repeated	Array of statistics values, each corresponding to a particular measurement, such as max, min, mean, etc

RegisterViewer

Source file: control/register_viewer.proto

REGISTER_VIEWER: Registers the viewer with the backend. Responds with *REGISTER_VIEWER_ACK*

Field	Type	Label	Description
session_id	fixed32		Unique session ID parameter (can be generated using UUID libraries). Passing in an existing session ID can be used for resuming sessions
api_key	string		Optional user-specific API key to be used for basic authentication. Could be an encrypted JWT for secure authentication.
client_feature_flags	fixed32		Optional feature bitflag specifying client-side feature set

RegisterViewerAck

Source file: control/register_viewer.proto

REGISTER_VIEWER_ACK Acknowledgement response for *REGISTER_VIEWER*. Informs the frontend whether the session was correctly.

Field	Type	Label	Description
session_id	fixed32		Unique session ID
success	bool		Defines whether the *REGISTER_VIEWER* command was successful
message	string		Error message (if applicable)
session_type	*SessionType*		Defines the type of session established
server_feature_flags	fixed32		Optional feature bitflag specifying server-side feature set
user_preferences	map<key: string, value: string>	repeated	Map of user preferences retrieved from the server database. If this is empty and the server does not have the USER_PREFERENCES feature flag set, then the user preferences are read from localStorage instead.
user_layouts	map<key: string, value: string>	repeated	Map of user layouts retrieved from the server database
platform_strings	map<key: string, value: string>	repeated	Map of server-generated platform information strings

RemoveRegion

Source file: control/region.proto

REMOVE_REGION: Removes a region

Field	Type	Label	Description
region_id	sfixed32		Unique region ID of the region to be removed

RemoveRequiredTiles

Source file: control/tiles.proto

REMOVE_REQUIRED_TILES Provides a list of tiles that are required for the specified file

Field	Type	Label	Description
file_id	sfixed32		The file ID that the view corresponds to
tiles	sfixed32	repeated	The list of tiles required, in encoded coordinate

ResumeSession

Source file: control/resume_session.proto

Field	Type	Label	Description
images	*ImageProperties*	repeated	
catalog_files	*OpenCatalogFile*	repeated	

ResumeSessionAck

Source file: control/resume_session.proto

Field	Type	Label	Description
success	bool		
message	string		

SaveFile

Source file: request/save_file.proto

Field	Type	Label	Description
file_id	sfixed32		
output_file_directory	string		
output_file_name	string		
output_file_type	*FileType*		The format of a new image file
region_id	sfixed32		
channels	sfixed32	repeated	Set image channels: [start, end, stride]
stokes	sfixed32	repeated	Set image stokes: [start, end, stride]
keep_degenerate	bool		
rest_freq	double		Set the rest frequency (Hz) of the image

SaveFileAck

Source file: request/save_file.proto

Field	Type	Label	Description
file_id	sfixed32		
success	bool		
message	string		

ScriptingRequest

Source file: request/scripting.proto

Field	Type	Label	Description
scripting_request_id	sfixed32		Used to connect a single scripting request to its response
target	string		the path of the target object. e.g. activeFrame.renderConfig
action	string		the name of the function to call. e.g. setColorMap
parameters	string		JSON array of parameters. e.g. '["viridis"]'
async	bool		flag indicating whether the frontend should execute this asynchronously, or only return once the call is complete
return_path	string		optional string indicating the path of the response sub-object to return. If this is empty, the entire response will be returned.

ScriptingResponse

Source file: request/scripting.proto

Field	Type	Label	Description
scripting_request_id	sfixed32		should match the incoming request ID
success	bool		indicates whether the call was correctly executed
message	string		optional error message
response	string		JSON-parsable response. e.g. "true", or the base64-encoded string

SetContourParameters

Source file: control/contour.proto

SET_CONTOUR_PARAMETERS Sets the contour parameters for a file

Field	Type	Label	Description
file_id	fixed32		The file ID that the contour corresponds to
reference_file_id	fixed32		The file ID of the reference image that the contour vertices should be mapped to
image_bounds	*ImageBounds*		The XY bounds corresponding to the image data in pixel coordinates
levels	double	repeated	Contour levels
smoothing_mode	*SmoothingMode*		Pre-contouring smoothing mode
smoothing_factor	int32		Contour smoothness factor. For block averaging, this is the block width For Gaussian smoothing, this defines both the Gaussian width, and the kernel size
decimation_factor	int32		Decimation factor, indicates to what 1/Nth of a pixel the contour vertices should be rounded to
compression_level	int32		Zstd compression level
contour_chunk_size	int32		Size of contour chunks, in number of vertices. If this is set to zero, partial contour results are not used

SetCursor

Source file: control/set_cursor.proto

SET_CURSOR: Sets the current cursor position in image space coordinates. The cursor defines a special case of a region, with a single control point.

Field	Type	Label	Description
file_id	sfixed32		Which file slot the cursor is moving over
point	*Point*		XY-coordinates of cursor in image space
spatial_requirements	*SetSpatialRequirements*		Optional accompanying spatial requirements message to be processed prior to cursor update

SetHistogramRequirements

Source file: control/region_requirements.proto

SET_HISTOGRAM_REQUIREMENTS: Sets which histogram data needs to be streamed to the frontend when the region is updated

Field	Type	Label	Description
file_id	sfixed32		Which file slot the requirements describe
region_id	sfixed32		ID of the region that is having requirements defined. If a region ID of -1 is given, this corresponds to the entire 2D image.
histograms	*HistogramConfig*	repeated	List of required histograms, along with the number of bins. If the channel is -1, the current channel is used. If the channel is -2, the histogram is constructed over all channels. If the number of bins is less than zero, an automatic bin size is used, based on the number of values.

SetImageChannels

Source file: control/set_image_channels.proto

SET_IMAGE_CHANNELS Sets the current image channel and Stokes parameter

Field	Type	Label	Description
file_id	sfixed32		The file ID that the view corresponds to
channel	sfixed32		The image channel (Z-coordinate)
stokes	sfixed32		The image stokes parameter
required_tiles	*AddRequiredTiles*		Required tiles when changing channels

SetRegion

Source file: control/region.proto

SET_REGION: Creates or updates a region. Backend responds with *SET_REGION_ACK*

Field	Type	Label	Description
file_id	sfixed32		File slot of the reference image
region_id	sfixed32		Unique region ID. <=0 if a new region is being created.
region_info	*RegionInfo*		Region parameters

SetRegionAck

Source file: control/region.proto

SET_REGION_ACK: Response for *SET_REGION*

Field	Type	Label	Description
success	bool		Defines whether *SET_REGION* was successful
message	string		Error message (if applicable)
region_id	sfixed32		The unique region ID. If the region is updated, this will be the same as the region ID specified in *SET_REGION*. If a new region is being created, the ID of the new region will be passed back.

SetSpatialRequirements

Source file: control/region_requirements.proto

SET_SPATIAL_REQUIREMENTS: Sets which information needs to be streamed to the frontend when the region is updated

Field	Type	Label	Description
file_id	sfixed32		Which file slot the requirements describe
region_id	sfixed32		ID of the region that is having requirements defined. If a region ID of 0 is given, this corresponds to the point region defined by the cursor position.
spatial_profiles	*SpatialConfig*	repeated	List of spatial profiles needed.

SetSpectralRequirements

Source file: control/region_requirements.proto

SET_SPECTRAL_REQUIREMENTS: Sets which spectral profile data needs to be streamed to the frontend when the region is updated

Field	Type	Label	Description
file_id	sfixed32		Which file slot the requirements describe
region_id	sfixed32		ID of the region that is having requirements defined. If a region ID of 0 is given, this corresponds to the point region defined by the cursor position.
spectral_profiles	*SpectralConfig*	repeated	List of spectral profiles needed, along with which stats types are needed for each profile.

SetStatsRequirements

Source file: control/region_requirements.proto

SET_STATS_REQUIREMENTS: Sets which stats data needs to be streamed to the frontend when the region is updated

Field	Type	Label	Description
file_id	sfixed32		Which file slot the requirements describe
region_id	sfixed32		ID of the region that is having requirements defined. If a region ID of -1 is given, this corresponds to the entire 2D image.
stats_configs	*StatsConfig*	repeated	List of required stats

SetVectorOverlayParameters

Source file: control/vector_overlay.proto

SET_VECTOR_OVERLAY_PARAMETERS Sets the overlay parameters for a file

Field	Type	Label	Description
file_id	fixed32		The file ID that the overlay corresponds to
image_bounds	*ImageBounds*		The XY bounds corresponding to the image data in pixel coordinates. Currently unused
smoothing_factor	fixed32		Block smoothing factor to use. Must be an even integer, corresponds to the `mip` coordinate.
fractional	bool		Whether to use fractional polarization intensity
threshold	double		Threshold value to use. If this is set to NaN, no threshold is applied.
debiasing	bool		Whether to use debiasing
q_error	double		Stokes Q error when debiasing
u_error	double		Stokes U error when debiasing
stokes_intensity	sfixed32		The Stokes coordinate to use when generating vector intensity. If this is < 0, uniform intensity is used. If both this and `stokes_angle` are < 0, the overlay requirement is cleared
stokes_angle	sfixed32		The Stokes coordinate to use when generating vector angle. If this is < 0, uniform angle is used (e.g. when rendering block markers)
compression_type	*CompressionType*		The compression algorithm to use.
compression_quality	float		Compression quality switch

SpatialConfig

Source file: control/region_requirements.proto

Field	Type	Label	Description
coordinate	string		The required spatial coordinate ("x" or "y").
start	sfixed32		The start of the required range (inclusive). If the start and end are the same (i.e. the range is empty), the default of 0 is used.
end	sfixed32		The end of the required range (exclusive). If the start and end are the same (i.e. the range is empty), the height or width of the image is used.
mip	sfixed32		The maximum required mip. The backend must return data of at least this resolution, but may return a higher resolution. If this is unset or 0, the full-resolution data is used.

SpatialProfileData

Source file: stream/spatial_profile.proto

SPATIAL_PROFILE_DATA: Data for spatial profile set for a specific file

Field	Type	Label	Description
file_id	sfixed32		The file ID that the profile corresponds to
region_id	sfixed32		The region_id corresponding to this profile. If the profile corresponds to the cursor position, the region ID is zero.
x	sfixed32		The pixel X-coordinate of the profile set
y	sfixed32		The pixel Y-coordinate of the profile set
channel	sfixed32		The image channel used to generate the profiles
stokes	sfixed32		The image stokes parameter used to generate the profiles
value	float		The value of the image at the given coordinates
profiles	*SpatialProfile*	repeated	Spatial profiles for each required profile type

SpectralConfig

Source file: control/region_requirements.proto

Field	Type	Label	Description
coordinate	string		The required spectral coordinate ("z"), optionally preceded by a polarization parameter. If no polarization parameter is present, or if the coordinate is empty, the active polarization parameter is used.
stats_types	*StatsType*	repeated	The required stats type. If the region is a point region, this field is ignored.

SpectralLineRequest

Source file: control/spectral_line_request.proto

Field	Type	Label	Description
frequency_range	*DoubleBounds*		
line_intensity_lower_limit	double		

SpectralLineResponse

Source file: control/spectral_line_request.proto

Field	Type	Label	Description
success	bool		
message	string		
data_size	sfixed32		
headers	*CatalogHeader*	repeated	
spectral_line_data	map<key: fixed32, value: *ColumnData*>	repeated	

SpectralProfileData

Source file: stream/spectral_profile.proto

SPECTRAL_PROFILE_DATA: Data for spectral profile set for a specific file

Field	Type	Label	Description
file_id	sfixed32		The file ID that the profile corresponds to
region_id	sfixed32		The region ID that the stats data corresponds to. If the profile corresponds to the cursor position, the region ID has a value of 0.
stokes	sfixed32		The image stokes parameter used to generate the profiles
progress	float		Progress indicator, in the case of partial profile results being sent. If the profile calculations are time-consuming, regular updates should be sent to the frontend. If the data is complete, progress >= 1.
profiles	*SpectralProfile*	repeated	Spatial profiles for each required profile type

SplataloguePing

Source file: control/spectral_line_request.proto

SplataloguePong

Source file: control/spectral_line_request.proto

Field	Type	Label	Description
success	bool		
message	string		

StartAnimation

Source file: control/animation.proto

START_ANIMATION: Starts an animation, as defined by the start, stop and step definitions. Backend responds with *START_ANIMATION_ACK*

Field	Type	Label	Description
file_id	sfixed32		Which file slot the animation describes.
first_frame	*AnimationFrame*		The lower bound of the animation when looping.
start_frame	*AnimationFrame*		The starting point of the animation.
last_frame	*AnimationFrame*		The upper bound of the animation.
delta_frame	*AnimationFrame*		The frame change step for the animation. For example, a delta frame of {channel=1, stokes=0} would step through each channel in the file.
frame_rate	sfixed32		Frame rate per second
looping	bool		Whether to loop the animation indefinitely.
reverse	bool		Whether to reverse the animation direction when endFrame is reached.
required_tiles	*AddRequiredTiles*		Required tiles when changing channels
matched_frames	map<key: sfixed32, value: *MatchedFrameList*>	repeated	
stokes_indices	sfixed32	repeated	Required stokes frames with respect to stokes types

StartAnimationAck

Source file: control/animation.proto

START_ANIMATION_ACK Response for *START_ANIMATION*

Field	Type	Label	Description
success	bool		Defines whether *START_ANIMATION* was successful
message	string		Error message (if applicable)
animation_id	sfixed32		The animation ID of the new animation

StatsConfig

Source file: control/region_requirements.proto

Field	Type	Label	Description
coordinate	string		
stats_types	*StatsType*	repeated	

StokesFile

Source file: control/concat_stokes_files.proto

Field	Type	Label	Description
directory	string		Required directory name
file	string		Required file name
hdu	string		Which HDU to load (if applicable). If left blank, the first HDU will be used
polarization_type	*PolarizationType*		Polarization type

StopAnimation

Source file: control/animation.proto

STOP_ANIMATION Stops the playing animation

Field	Type	Label	Description
file_id	sfixed32		Which file slot the animation describes.
end_frame	*AnimationFrame*		The ending point of the animation.

StopFileList

Source file: request/file_list.proto

Field	Type	Label	Description
file_list_type	*FileListType*		

StopMomentCalc

Source file: control/stop_moment_calc.proto

Field	Type	Label	Description
file_id	sfixed32		Stop the moment calculation with respect to the image file id

StopPvCalc

Source file: control/stop_pv_calc.proto

Field	Type	Label	Description
file_id	sfixed32		Stop the PV image calculation with respect to the image file id

VectorOverlayTileData

Source file: stream/vector_overlay_tile.proto

Field	Type	Label	Description
file_id	sfixed32		The file ID that the vector overlay image corresponds to
channel	sfixed32		The image channel (z-coordinate)
stokes_intensity	sfixed32		The Stokes coordinate that was used to generate vector intensity. If this is < 0, uniform intensity is used
stokes_angle	sfixed32		The Stokes coordinate that was used to generate vector angle. If this is < 0, uniform angle is used (e.g. when rendering block markers)
compression_type	*CompressionType*		The compression algorithm used.
compression_quality	float		Compression quality switch
intensity_tiles	*TileData*	repeated	List of tile data for vector intensity. The length of this list must match that of `angle_tiles`, or be zero
angle_tiles	*TileData*	repeated	List of tile data for vector angle. The length of this list must match that of `intensity_tiles`, or be zero
progress	double		Progress of the vector overlay being sent. If this is zero, the message is assumed to contain the entire contour sets

2.5.2 Sub-messages

AnimationFrame

Source file: shared/defs.proto

Field	Type	Label	Description
channel	sfixed32		
stokes	sfixed32		

Beam

Source file: shared/defs.proto

describe each beam for multi-beam images

Field	Type	Label	Description
channel	sfixed32		
stokes	sfixed32		
major_axis	float		
minor_axis	float		
pa	float		

CatalogFileInfo

Source file: shared/defs.proto

Field	Type	Label	Description
name	string		
type	*CatalogFileType*		
file_size	sfixed64		
description	string		
coosys	*Coosys*	repeated	
date	sfixed64		

CatalogHeader

Source file: shared/defs.proto

Field	Type	Label	Description
name	string		
data_type	*ColumnType*		
column_index	sfixed32		
description	string		
units	string		

CatalogImageBounds

Source file: shared/defs.proto

Field	Type	Label	Description
x_column_name	string		
y_column_name	string		
image_bounds	*ImageBounds*		

ColumnData

Source file: shared/defs.proto

Field	Type	Label	Description
data_type	*ColumnType*		
string_data	string	repeated	All data types other than string sent as binary
binary_data	bytes		binary data will get converted to a TypedArray

Coosys

Source file: shared/defs.proto

Field	Type	Label	Description
equinox	string		
epoch	string		
system	string		

DirectoryInfo

Source file: shared/defs.proto

Directory info message structure (internal use only)

Field	Type	Label	Description
name	string		
item_count	sfixed32		
date	sfixed64		

DoubleBounds

Source file: shared/defs.proto

Field	Type	Label	Description
min	double		
max	double		

DoublePoint

Source file: shared/defs.proto

Field	Type	Label	Description
x	double		
y	double		

FileInfo

Source file: shared/defs.proto

File info message structure (internal use only)

Field	Type	Label	Description
name	string		
type	*FileType*		
size	sfixed64		
HDU_list	string	repeated	
date	sfixed64		

FileInfoExtended

Source file: shared/defs.proto

Field	Type	Label	Description
dimensions	sfixed32		Number of dimensions of the image file
width	sfixed32		Width of the XY plane
height	sfixed32		Height of the XY plane
depth	sfixed32		Number of channels
stokes	sfixed32		Number of Stokes parameters
stokes_vals	string	repeated	List of Stokes parameters contained in the file (if applicable). For files that do not explicitly specify Stokes files, this will be blank.
header_entries	*HeaderEntry*	repeated	Header entries from header string or attributes
computed_entries	*HeaderEntry*	repeated	

FilterConfig

Source file: shared/defs.proto

Field	Type	Label	Description
column_name	string		
comparison_operator	*ComparisonOperator*		
value	double		
secondary_value	double		
sub_string	string		

FloatBounds

Source file: shared/defs.proto

Field	Type	Label	Description
min	float		
max	float		

GaussianComponent

Source file: shared/defs.proto

parameters of a 2D Gaussian component for image fitting

Field	Type	Label	Description
center	*DoublePoint*		x/y coordinate of the center in pixels
amp	double		amplitude of the component
fwhm	*DoublePoint*		full width at half maximum along x/y coordinate in pixels
pa	double		position angle in degrees

HeaderEntry

Source file: shared/defs.proto

Field	Type	Label	Description
name	string		
value	string		
entry_type	*EntryType*		
numeric_value	double		
comment	string		

Histogram

Source file: shared/defs.proto

Field	Type	Label	Description
num_bins	sfixed32		
bin_width	double		
first_bin_center	double		
bins	sfixed32	repeated	
mean	double		
std_dev	double		

ImageBounds

Source file: shared/defs.proto

Field	Type	Label	Description
x_min	sfixed32		
x_max	sfixed32		
y_min	sfixed32		
y_max	sfixed32		

IntBounds

Source file: shared/defs.proto

Field	Type	Label	Description
min	sfixed32		
max	sfixed32		

ListProgress

Source file: shared/defs.proto

Field	Type	Label	Description
file_list_type	*FileListType*		
percentage	float		
checked_count	sfixed32		
total_count	sfixed32		

MatchedFrameList

Source file: shared/defs.proto

Field	Type	Label	Description
frame_numbers	float	repeated	

Point

Source file: shared/defs.proto

Field	Type	Label	Description
x	float		
y	float		

RegionInfo

Source file: shared/defs.proto

Field	Type	Label	Description
region_type	*RegionType*		The type of region described by the control points. The meaning of the control points will differ, depending on the type of region being defined.
control_points	*Point*	repeated	Control points for the region
rotation	float		(Only applicable for ellipse and rectangle) Rotation of the region in the xy plane (radians).

RegionStyle

Source file: shared/defs.proto

Field	Type	Label	Description
name	string		The name of the region, displayed as an annotation label.
color	string		Color as a name ("blue"), RGB string, or hex string
line_width	sfixed32		Width in pixels
dash_list	sfixed32	repeated	Dash length: on, off

SpatialProfile

Source file: shared/defs.proto

Field	Type	Label	Description
start	sfixed32		
end	sfixed32		
raw_values_fp32	bytes		
coordinate	string		
mip	sfixed32		

SpectralProfile

Source file: shared/defs.proto

Field	Type	Label	Description
coordinate	string		
stats_type	*StatsType*		
raw_values_fp32	bytes		
raw_values_fp64	bytes		

StatisticsValue

Source file: shared/defs.proto

Field	Type	Label	Description
stats_type	*StatsType*		
value	double		

TileData

Source file: shared/defs.proto

Field	Type	Label	Description
layer	sfixed32		Tile layer coordinate. If this is < 0, the `mip` value is used for coordinates instead
x	sfixed32		Tile x coordinate
y	sfixed32		Tile y coordinate
width	sfixed32		Width of the tile data. If this is left as zero, the default tile size should be used
height	sfixed32		Height of the tile data. If this is left as zero, the default tile size should be used
image_data	bytes		Image data. For uncompressed data, this is converted into FP32, while for compressed data, this is passed to the compression library for decompression.
nan_encodings	bytes		Run-length encodings of NaN values. These values are used to restore the NaN values after decompression.
mip	sfixed32		Mip coordinate. Ignored if `layer` >= 0

2.5.3 Enums

CatalogFileType

Source file: shared/enums.proto

Name	Number	Description
FITSTable	0	
VOTable	1	
Unknown	2	

ClientFeatureFlags

Source file: shared/enums.proto

Name	Number	Description
CLIENT_FEATURE_NONE	0	
WEB_GL	1	
WEB_GL_2	2	
WEB_ASSEMBLY	4	
WEB_ASSEMBLY_THREADS	8	
OFFSCREEN_CANVAS	16	

ColumnType

Source file: shared/enums.proto

Name	Number	Description
UnsupportedType	0	
String	1	
Uint8	2	
Int8	3	
Uint16	4	
Int16	5	
Uint32	6	
Int32	7	
Uint64	8	
Int64	9	
Float	10	
Double	11	
Bool	12	

ComparisonOperator

Source file: shared/enums.proto

Name	Number	Description
Equal	0	
NotEqual	1	
Lesser	2	
Greater	3	
LessorOrEqual	4	
GreaterOrEqual	5	
RangeOpen	6	
RangeClosed	7	

CompressionType

Source file: shared/enums.proto

Name	Number	Description
NONE	0	
ZFP	1	
SZ	2	

CoordinateType

Source file: shared/enums.proto

Name	Number	Description
PIXEL	0	
WORLD	1	

EntryType

Source file: shared/enums.proto

Name	Number	Description
STRING	0	
FLOAT	1	
INT	2	

ErrorSeverity

Source file: shared/enums.proto

Name	Number	Description
DEBUG	0	
INFO	1	
WARNING	2	
ERROR	3	
CRITICAL	4	

EventType

Source file: shared/enums.proto

Name	Number	Description
EMPTY_EVENT	0	
REGISTER_VIEWER	1	
FILE_LIST_REQUEST	2	
FILE_INFO_REQUEST	3	
OPEN_FILE	4	

continues on next page

Table 1 – continued from previous page

Name	Number	Description
SET_IMAGE_CHANNELS	6	
SET_CURSOR	7	
SET_SPATIAL_REQUIREMENTS	8	
SET_HISTOGRAM_REQUIREMENTS	9	
SET_STATS_REQUIREMENTS	10	
SET_REGION	11	
REMOVE_REGION	12	
CLOSE_FILE	13	
SET_SPECTRAL_REQUIREMENTS	14	
START_ANIMATION	15	
START_ANIMATION_ACK	16	
STOP_ANIMATION	17	
REGISTER_VIEWER_ACK	18	
FILE_LIST_RESPONSE	19	
FILE_INFO_RESPONSE	20	
OPEN_FILE_ACK	21	
SET_REGION_ACK	22	
REGION_HISTOGRAM_DATA	23	
SPATIAL_PROFILE_DATA	25	
SPECTRAL_PROFILE_DATA	26	
REGION_STATS_DATA	27	
ERROR_DATA	28	
ANIMATION_FLOW_CONTROL	29	
ADD_REQUIRED_TILES	30	
REMOVE_REQUIRED_TILES	31	
RASTER_TILE_DATA	32	
REGION_LIST_REQUEST	33	
REGION_LIST_RESPONSE	34	
REGION_FILE_INFO_REQUEST	35	
REGION_FILE_INFO_RESPONSE	36	
IMPORT_REGION	37	
IMPORT_REGION_ACK	38	
EXPORT_REGION	39	
EXPORT_REGION_ACK	40	
SET_CONTOUR_PARAMETERS	45	
CONTOUR_IMAGE_DATA	46	
RESUME_SESSION	47	
RESUME_SESSION_ACK	48	
RASTER_TILE_SYNC	49	
CATALOG_LIST_REQUEST	50	
CATALOG_LIST_RESPONSE	51	
CATALOG_FILE_INFO_REQUEST	52	
CATALOG_FILE_INFO_RESPONSE	53	
OPEN_CATALOG_FILE	54	
OPEN_CATALOG_FILE_ACK	55	
CLOSE_CATALOG_FILE	56	
CATALOG_FILTER_REQUEST	57	
CATALOG_FILTER_RESPONSE	58	
SCRIPTING_REQUEST	59	

continues on next page

Table 1 – continued from previous page

Name	Number	Description
SCRIPTING_RESPONSE	60	
MOMENT_REQUEST	61	
MOMENT_RESPONSE	62	
MOMENT_PROGRESS	63	
STOP_MOMENT_CALC	64	
SAVE_FILE	65	
SAVE_FILE_ACK	66	
SPECTRAL_LINE_REQUEST	67	
SPECTRAL_LINE_RESPONSE	68	
CONCAT_STOKES_FILES	69	
CONCAT_STOKES_FILES_ACK	70	
FILE_LIST_PROGRESS	71	
STOP_FILE_LIST	72	
SPLATALOGUE_PING	73	
SPLATALOGUE_PONG	74	
PV_REQUEST	75	
PV_RESPONSE	76	
PV_PROGRESS	77	
STOP_PV_CALC	78	
FITTING_REQUEST	79	
FITTING_RESPONSE	80	
SET_VECTOR_OVERLAY_PARAMETERS	81	
VECTOR_OVERLAY_TILE_DATA	82	

FileFeatureFlags

Source file: shared/enums.proto

Name	Number	Description
FILE_FEATURE_NONE	0	
ROTATED_DATASET	1	
CHANNEL_HISTOGRAMS	2	
CUBE_HISTOGRAMS	4	
CHANNEL_STATS	8	
MEAN_IMAGE	16	
MIP_DATASET	32	

FileListFilterMode

Source file: shared/enums.proto

Name	Number	Description
Content	0	
Extension	1	
AllFiles	2	

FileListType

Source file: shared/enums.proto

Name	Number	Description
Image	0	
Catalog	1	

FileType

Source file: shared/enums.proto

Name	Number	Description
CASA	0	
CRTF	1	
DS9_REG	2	
FITS	3	
HDF5	4	
MIRIAD	5	
UNKNOWN	6	

Moment

Source file: shared/enums.proto

Name	Number	Description
MEAN_OF_THE_SPECTRUM	0	
INTE-GRATED_OF_THE_SPECTRUM	1	
INTEN-SITY_WEIGHTED_COORD	2	
INTEN-SITY_WEIGHTED_DISPERSION_OF_THE_COORD	3	
MEDIAN_OF_THE_SPECTRUM	4	
MEDIAN_COORDINATE	5	
STD_ABOUT_THE_MEAN_OF_THE_SPECTRUM	6	
RMS_OF_THE_SPECTRUM	7	
ABS_MEAN_DEVIATION_OF_THE_SPECTRUM	8	
MAX_OF_THE_SPECTRUM	9	
CO-ORD_OF_THE_MAX_OF_THE_SPECTRUM	10	
MIN_OF_THE_SPECTRUM	11	
CO-ORD_OF_THE_MIN_OF_THE_SPECTRUM	12	

MomentAxis

Source file: shared/enums.proto

Name	Number	Description
SPECTRAL	0	
STOKES	1	

MomentMask

Source file: shared/enums.proto

Name	Number	Description
None	0	
Include	1	
Exclude	2	

PolarizationType

Source file: shared/enums.proto

polarization parameters including the Stokes parameters, circular correlations, and linear correlations (the Stokes axis defined by the FITS standard)

Name	Number	Description
POLARIZATION_TYPE_NONE	0	
I	1	
Q	2	
U	3	
V	4	
RR	5	
LL	6	
RL	7	
LR	8	
XX	9	
YY	10	
XY	11	
YX	12	
Ptotal	13	Polarized intensity: sqrt(Q^2 + U^2 + V^2)
Plinear	14	Linearly Polarized intensity: sqrt(Q^2 + U^2)
PFtotal	15	Polarization Fraction: Ptotal / I
PFlinear	16	Linear Polarization Fraction: Plinear / I
Pangle	17	Linear Polarization Angle: arctan(U/Q) / 2

RegionType

Source file: shared/enums.proto

Name	Number	Description
POINT	0	
LINE	1	
POLYLINE	2	
RECTANGLE	3	
ELLIPSE	4	
ANNULUS	5	
POLYGON	6	

RenderMode

Source file: shared/enums.proto

Name	Number	Description
RASTER	0	
CONTOUR	1	

ServerFeatureFlags

Source file: shared/enums.proto

Name	Number	Description
SERVER_FEATURE_NONE	0	
SZ_COMPRESSION	1	
HEVC_COMPRESSION	2	
NVENC_COMPRESSION	4	
READ_ONLY	8	Disables write requests, including saving files, exporting regions, and writing preferences and layouts files.
USER_PREFERENCES	16	
USER_LAYOUTS	32	
SCRIPTING	64	

SessionType

Source file: shared/enums.proto

Name	Number	Description
NEW	0	
RESUMED	1	

SmoothingMode

Source file: shared/enums.proto

Name	Number	Description
NoSmoothing	0	
BlockAverage	1	
GaussianBlur	2	

SortingType

Source file: shared/enums.proto

Name	Number	Description
Ascending	0	
Descending	1	

StatsType

Source file: shared/enums.proto

Name	Number	Description
NumPixels	0	
NanCount	1	
Sum	2	
FluxDensity	3	
Mean	4	
RMS	5	
Sigma	6	
SumSq	7	
Min	8	
Max	9	
Extrema	10	
Blc	11	
Trc	12	
MinPos	13	
MaxPos	14	
Blcf	15	
Trcf	16	
MinPosf	17	
MaxPosf	18	