Drawing and Document Annotation Module

Developer's Guide

Abhijit Parate

Advisor

Dr. Barry Levine

Table of Contents

Drawing & Document Annotation Module

Overview

The drawing and document annotation module is developed to facilitate providers to record drawing and sketches and add additional contextual information to the drawings for example, images, audios, videos, etc. which can also be embedded in the drawings itself and link them to patient's medical records within OpenMRS.

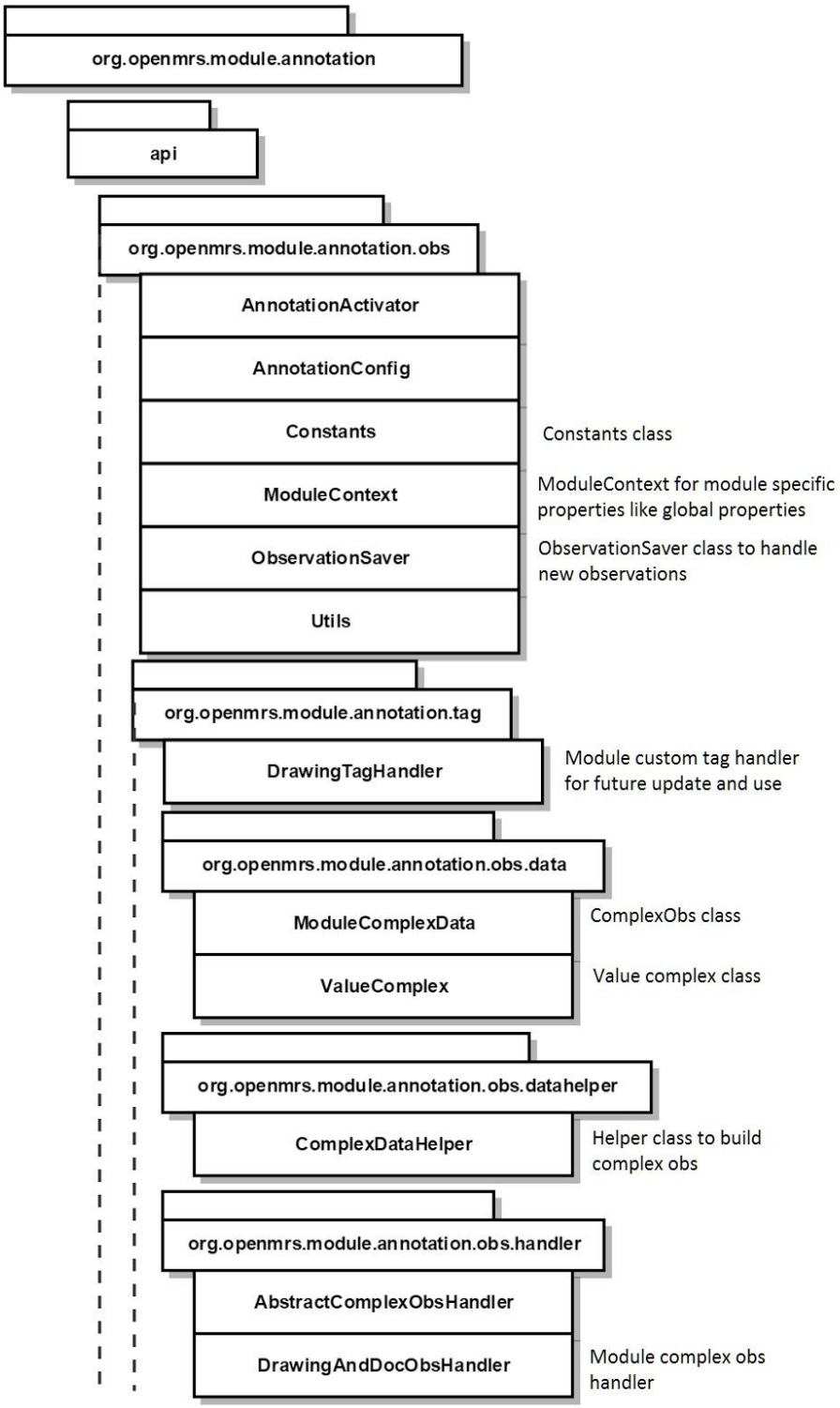
The primary element of interest of the module is the drawing editor. Being the most used part of the module, the editor provides user tools to draw on canvas efficiently and record additional contextual information.

This guide provides future developer with the necessary information regarding the module and its architecture to improve or develop further on the module.

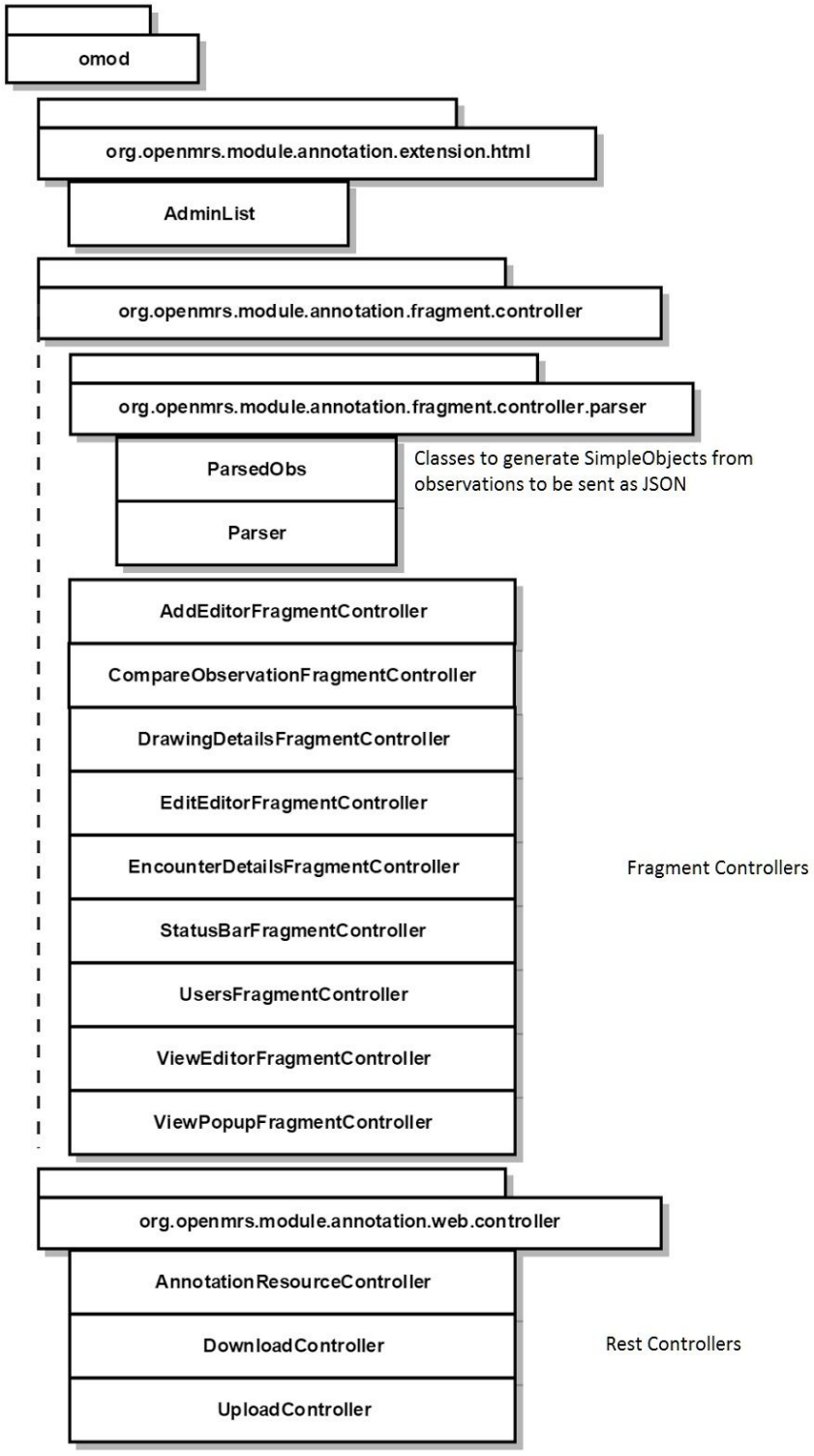
Module Package Structure

Project consists of two modules as follows

api


org.openmrs.module.annotation
api
org.openmrs.module.annotation.obs
AnnotationActivator
AnnotationConfig
Constants
Constants class
ModuleContext
ModuleContext for module specific properties like global properties
ObservationSaver
ObservationSaver class to handle new observations
Utils
org.openmrs.module.annotation.tag
DrawingTagHandler
Module custom tag handler for future update and use
org.openmrs.module.annotation.obs.data
ModuleComplexData
ComplexObs class
ValueComplex
Value complex class
org.openmrs.module.annotation.obs.datahelper
ComplexDataHelper
Helper class to build complex obs
org.openmrs.module.annotation.obs.handler
AbstractComplexObsHandler
DrawingAndDocObsHandler
Module complex obs handler

omod


omod
org.openmrs.module.annotation.extension.html
AdminList
org.openmrs.module.annotation.fragment.controller
org.openmrs.module.annotation.fragment.controller.parser
ParsedObs
Parser
Classes to generate SimpleObjects from observations to be sent as JSON
AddEditorFragmentController
CompareObservationFragmentController
DrawingDetailsFragmentController
EditEditorFragmentController
EncounterDetailsFragmentController
StatusBarFragmentController
UsersFragmentController
ViewEditorFragmentController
ViewPopupFragmentController
Fragment Controllers
org.openmrs.module.annotation.web.controller
AnnotationResourceController
DownloadController
UploadController
Rest Controllers

webapp

The webapp consists of pages and fragments

Fragments

1. *addDrawing.gsp* -
   Contains fragment to be added to Add Drawing page
2. *editDrawing.gsp* -
   Contains fragment to be added to Edit Drawing page
3. *viewDrawing.gsp* -
   Contains fragment to be added to View Drawing page
4. *viewDrawingPopup.gsp* -
   Contains fragment to be added to View Drawing in a popup page
5. *compareObs.gsp* -
   Contains fragment to be added to Compare Obs page
6. *includes/canvas.gsp* -
   This contains the canvas element and the buttons from zoom and pan section.
7. *includes/actions_panel.gsp* -
   This contains the left action panel for drawing tools.
8. *includes/modals.gsp* -
   This contains the modals used for pencil popover, shapes popover, etc
9. *includes/statusBar.gsp* -
   This contains the action buttons like Save, Cancel and Attachments.
10. *includes/header.gsp* -
    This includes the common includes for all the pages.

Pages

1. *addDrawing.gsp* -
   Page with title “Add Drawing”
2. *compareObs.gsp* -
   Page with title “Compare Observations”
3. *editDrawing.gsp* -
   Page with title “Edit Drawing”
4. *viewDrawing.gsp* -
   Page with title “View Drawing”
5. *viewDrawingPopup.gsp* -
   Page with title “View Drawing”

Resources

Contains the resource used in fragments and pages like images, javascripts and css.

1. Images
2. Scripts - Contains the JS files
    a. attachments

        i. attachment-audio.js - JS to process audio attachments
        ii. attachment-file.js - JS to process other file attachments
        iii. attachment-image.js - JS to process image attachments
        iv. attachment-notes.js - JS to process note or text file attachments
        v. attachment-video.js - JS to process video attachments
    b. libs - Contains the libraries and plugins
    c. actions.js - JS for the drawing tools
    d. attachment.js - JS to process and handle attachments
    e. canvas.js - JS for canvas element
    f. compareObs.js
    g. context_menu.js - JS for context menu for the canvas
    h. drawingObsEncounterTemplate.js - JS for encounter details template
    i. editor.js - JS for canvas actions like undo, redo, clear, etc.
    j. import.js - JS for importing images into canvas
    k. keyboard.js - JS to handle keyboard button press like Delete, arrow keys, etc
    l. popup.js - JS for pop-up view page
    m. status_actions.js - JS for actions in the status bar like Save, Cancel, etc
    n. ui.js
3. Style
    a. font - fonts for the lightcase library
    b. libs - css for the libraries
    c. attachment.css
    d. compareObs.css
    e. controls.css
    f. drawingObsEncounterTemplate.css
    g. editor.css

Rest Controllers

Save Observations

This is used to create new encounters. Attach all the files and necessary parameters to link them to the patient's records.

URL:

<root>/rest/v1/upload.form

Method:

POST

Data Params

Required:

- files [ ] - DataUrl of the files to be saved
- fileNames [ ] - FileNames in the same order as files
- patientid -
- visitid -
- providerid -

Optional:

- obs [ ] -

Success Response:

{ "result" : "success" }

Error Response:

{ "result" : "failed" }

Fetch Observations

USed for downloading the files for preview in the patients dashboard.

URL:

<root>/rest/v1/annotation/obs/{obsUuid}/{filename}

Method:

GET | *POST*

URL Params

Required:

- *obsUuid* - UUID of the observation
- *filename* - Filename of the observation

Success Response:

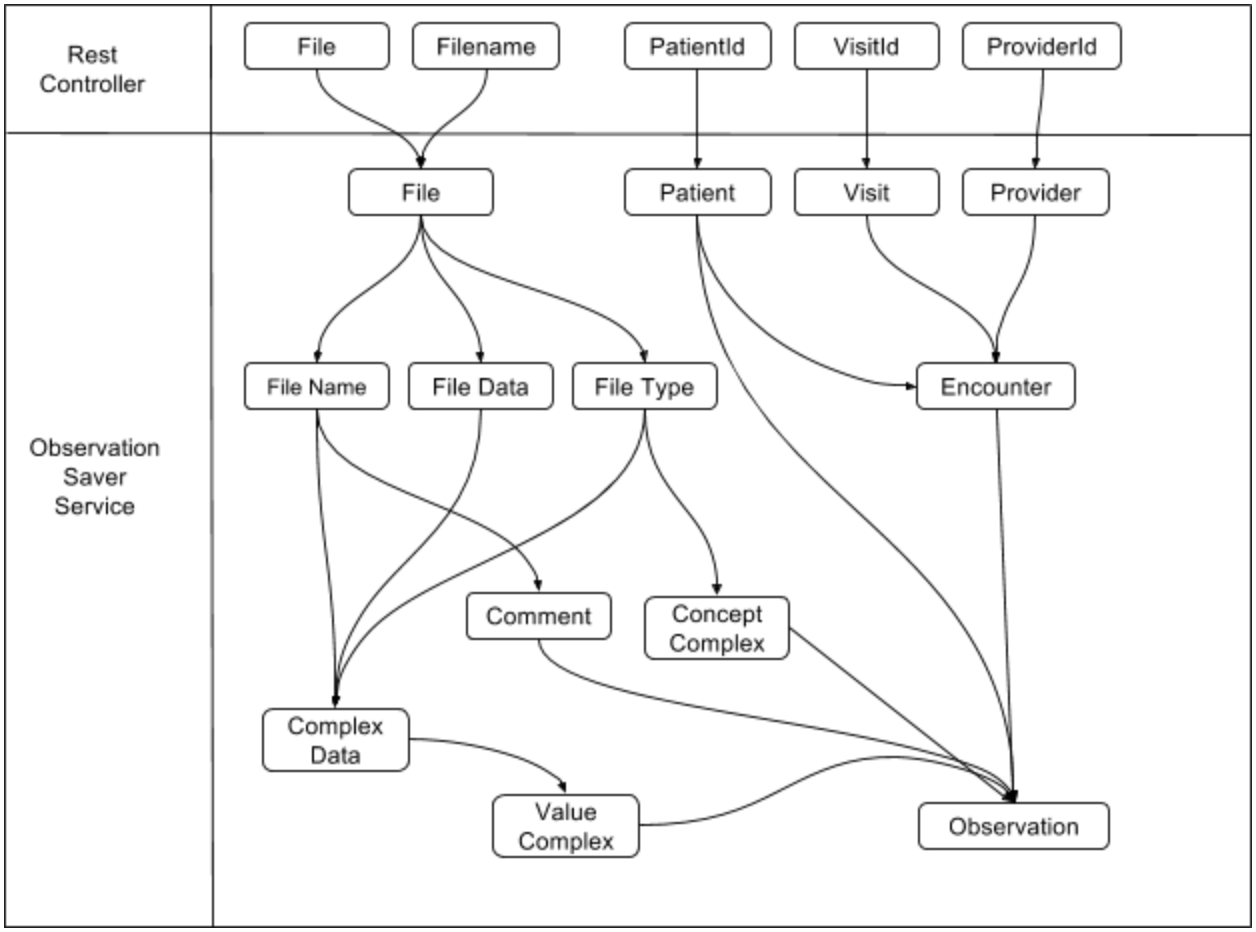
File

Success Response:

404 - Resource not found

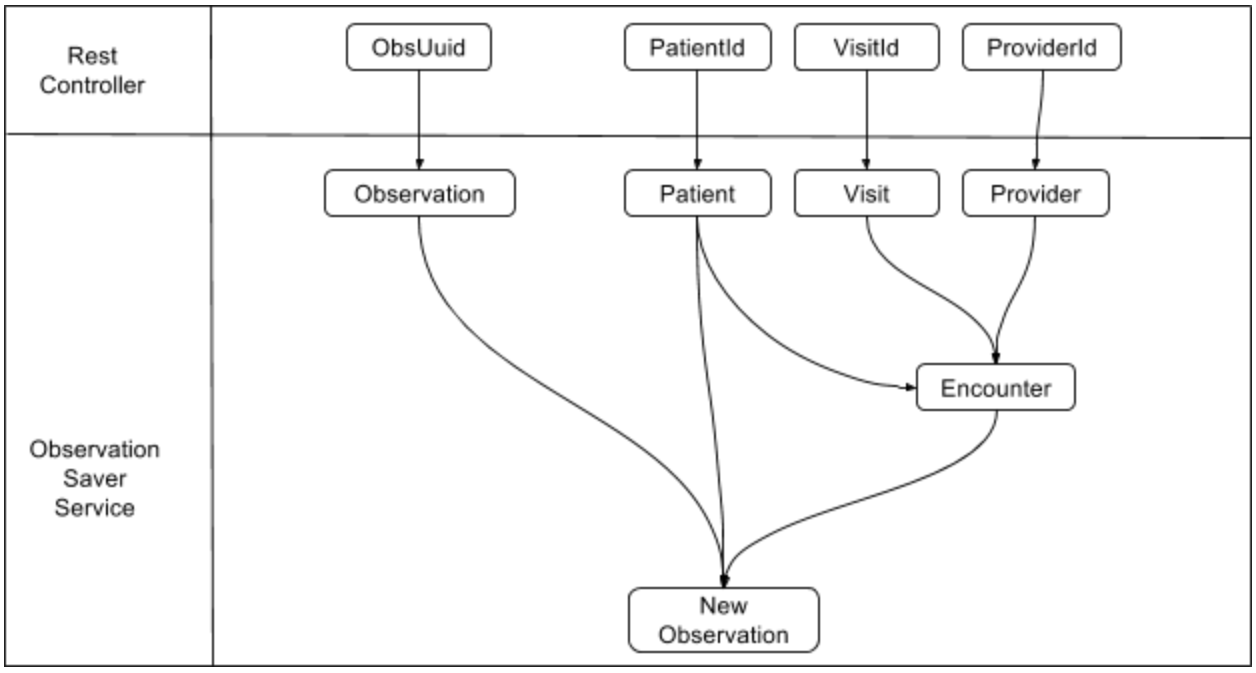
Creating Observations

From File



We save the file as a complex data and create a value complex to be persisted along with the observation. The file content type is used to create the appropriate concept complex which later is used to create the value complex.

From Previous Observations



We used the UUID of the previous observation to find it and the file. We clone the observation and save it as a new observation.

The complex observations are stored and linked to observation using the value complex. The value complex is stored in the following format

<Identifier> : <Content type> : <Mime-type> : <File name>

Identifier - used to identify if the observation if from our module

Content type - The type of the observation

The available types of observations are

- File
- SVG
- Image
- Video
- Audio
- Notes

Mime-type - mime type of the file

File-name - name of the file on server