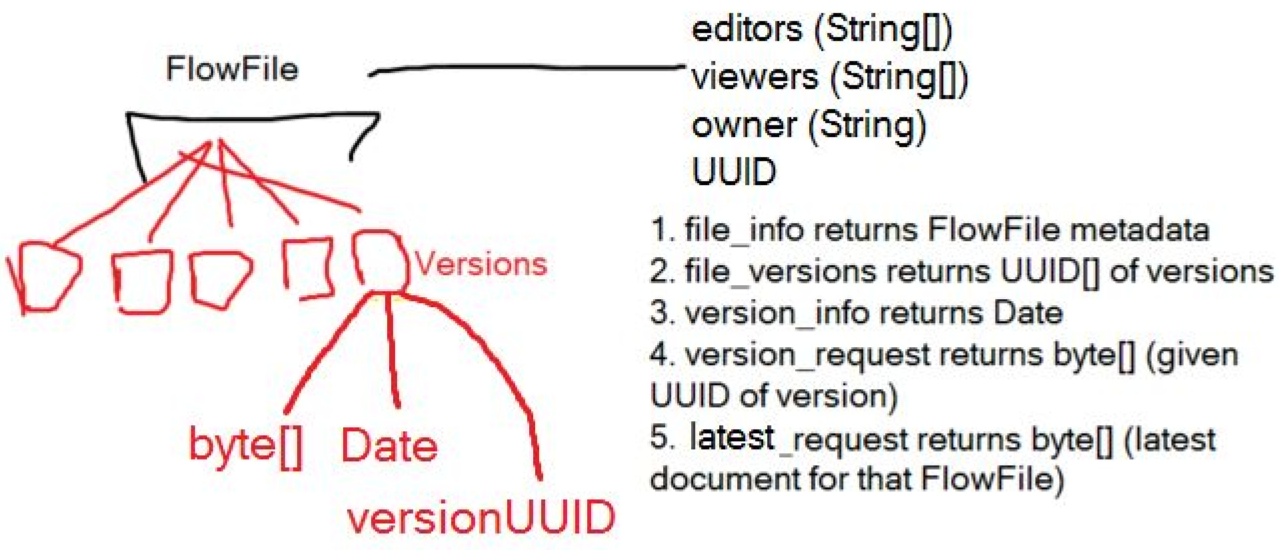

1. file_info returns FlowFile metadata
2. file_versions returns UUID[] of versions
3. version_info returns Date
4. version_request returns byte[] (given UUID of version)
5. latest_request returns byte[] (latest document for that FlowFile)

“Flow” Program Definition and Protocol

[Link to the Github repository]()

Abstract

There will be one central server and multiple clients, analogous to Google Docs. Each client may connect to the server, needing to provide a username and password in order to log into their session.
Each user has “projects” similar to Eclipse projects/IntelliJ modules/Github repositories etc. under their account, which they may choose to share with other select users. Each user will also have a list of projects that someone else has shared with them.
When a client opens a file inside of a project for editing, the client immediately notifies the server that they intend to edit the file. The exact protocol for this will function will be defined later in this document.

Client Connection Protocol

When the client connects to the server, the server will send back a list of projects that they own and a list of projects shared with them, along with the directory file structure of each project so the client will be able to browse files to select for editing.
If the client selects a file for editing (i.e. they click the file on the file directory panel), then the server will be notified that the client intends to edit the file. From this point onwards, the further protocol is defined in *Document Modification Protocol*.

User Modification Protocol

This protocol concerns when a user makes changes to their account. The following base operations are defined for a user, and are going to be in the client window’s settings menu:

- Creation of an account
- Password change
- Deletion of an account
- Adding/removing a project from the account

Project Modification Protocol
This protocol concerns when modifications are made to a project itself. The following base operations are defined for a project:

- Updating permissions for users a project is shared with
- Adding/removing files in a project
- Adding/removing directories in a project
- Renaming a project
- Deletion of a project

After any of these operations are made, the server must broadcast the operation to every client in order to update the graphical user interface.

Document Modification Protocol
This protocol concerns when the client makes textual changes to a document inside of a project. At this point, a client has selected a file for editing, and a display has appeared to the client which allows them to edit code. The server will keep track of the following properties of each client while browsing a project:

- The file they are looking at, or none
- The position of their cursor

When a client makes a change to the document, the base operation is sent to the server for broadcast to other clients. A list of base operations the client may execute are defined, similar to instructions in assembly. The base operations are following:

- Insertion of a character or selection
- Deletion of a character or selection
- Newline
- Change of cursor position

Each primitive operation also has an operational transform associated with it, in order to preserve cursor positions of other users. For example, Alice and Bob are connected to the same document. Alice is editing line 50 and Bob is editing line 100. Alice sends a “newline” operation at the end of line 50, therefore inserting a new line at that position. The operational transform associated with this operation would be to shift all the cursor positions of all other users down by one line, if the line of their cursor position is higher than where the operation happened.

Document Indexing
Each operation will be based on an index in the document and the size of the operation. New line characters are considered as a character, and will increment the index by one. For example, an example operation is defined below:

```
INSERT(60, “AAA”)
```

This will insert the character sequence “AAA” at line unknown and character position 60. Note that the operational transformation associated with this operation would be to shift all cursors after position 60 on line unknown by 3 character indices.

Data Storage Structure
All the data will be broken down into six tables: users, projects, directories, documents, access, and sessions. The columns inside each table are specified below:

- **Users**
    - Username (max 16 characters)
    - Password (Hashed)
- **Projects**
    - ProjectID (UUID associated with project)

    - ProjectName (character limit 16)
    - OwnerUsername
- **Directories**
    - DirectoryID (UUID associated with directory)
    - ParentDirectoryID (UUID associated with parent directory, if it exists)
    - DirectoryName
    - ProjectID (UUID associated with project)
- **Documents**
    - DocumentID (UUID associated with document)
    - ProjectID (UUID associated with project)
    - DocumentName (including extension, ex. ‘Client.xxx’)
    - ParentDirectoryID (UUID associated with parent directory)
- **Access**
    - ProjectID
    - Username
    - AccessLevel (NONE[0], VIEW[1], EDIT [2], OWNER [3])
- **Sessions**
    - Username
    - SessionID
- **Versions**
    - VersionID (UUID associated with directory)
    - Date (toString of the Date associated version)
    - DocumentID (UUID associated with document)

File Storage Structure (OUT OF DATE)
All the files will be stored in their original form (i.e. as a .java file) on the server-side. These files will be stored in directories based on the UUIDs of their respective owners / projects / directories.
For example, for a file ‘Client.java’, which is part of a project with a UUID ‘ABC’ which is owned by user “Billy”, and inside a directory with a UUID ‘ZEF’, the path to access the file could be: ‘data/Billy/ABC/ZEF/34AS-SDF8-SAS9-SD86/SDF6-GIH7-SDY7-SDFG’.
General Form: ‘data/Username/ProjectID/DirectoryID(s)/FileUUID/VersionUUID’

FULL COMMUNICATION PROTOCOL

REFER TO Flow-Commons Github FOR A DESCRIPTION OF SENDING MESSAGES

LOW LEVEL COMMUNICATION PROTOCOL [Implementation]

Will be using a system analogous to Hypertext Transfer Protocol.
Client sends HTTP GET request (Flow-Commons/Data) to server.
Server responds with HTTP response (Flow-Commons/Data) to client, where the “HTTP Status code” is the property ‘status’.
Connections are created instantaneously and then subsequently destroyed after data is transferred.

HIGH LEVEL COMMUNICATION PROTOCOL [FML]

The entire protocol is represented in “Flow Markup Language” (FML).
ALL* COMMANDS REQUIRE A **SESSION_ID**.
*Except for registering a new user

EXAMPLE OF FML:

```
@[DEPRECATED]
ACTION OR CONDITION OF DATA TRANSFER
{
        key:          “value”             type: <type>   ([Note <X>])
        # IF key = value
        [
                conditional_property:           ...     ...
        ]

        enum_key:       <enum_value>            type: String
        {
                POSSIBLE
                VALUES
                FOR
                KEY
        }

        (optional:)     "value>"                type: String

        # IF enum_key = POSSIBLE
        [
                conditional_property:           <OBJECT>        ...
                {
                        .objectMethod()         =>      TYPE    <OBJECT>        DESC    PROPERTY
                }
        ]
}
```

```
Prerequisite: CLIENT AND SERVER ESTABLISH TCP HANDSHAKE
```

:: USER LOGIN

```
Description: Client wants to log in.

CLIENT SENDS FLOW-COMMONS/DATA TO SERVER OF FORMAT
{
        type:           "login"                 type: String
        username:       <USERNAME>              type: String    [Note 1]
        password:       <PASSWORD>              type: String    [Note 2]
}
    1.   USERNAME is max 16 chars, 2 BYTE CHARS OF UTF-8
    2.   PASSWORD IS NOT HASHED

SERVER RESPONDS WITH FLOW-COMMONS/DATA TO CLIENT OF FORMAT
{
        status:         <STATUS>                type: String
        {
                OK
                USER_ALREADY_LOGGED_IN
                USERNAME_DOES_NOT_EXIST
                PASSWORD_INCORRECT
        }
        # IF status = OK else NULL
        [
                session_id:     <SESSION_ID>            type: UUID
        ]
}
```

:: USER LOGOUT

```
Description: Client wants to log in.

CLIENT SENDS FLOW-COMMONS/DATA TO SERVER OF FORMAT
{
        type:           "end_session"           type: String
        session_id:     <SESSION_ID>            type: UUID
}

SERVER RESPONDS WITH FLOW-COMMONS/DATA TO CLIENT OF FORMAT
```

```
{
        status:       <STATUS>             type: String
        {
                OK
                INVALID_SESSION_ID
        }
}
```

:: USER ACCOUNT MODIFICATION

Description: Client wants to modify their account.

```
CLIENT SENDS FLOW-COMMONS/DATA TO SERVER OF FORMAT
{
        type:         "user"               type: String
        user_type:    <USER_TYPE>          type: String
        {
                REGISTER
                CLOSE_ACCOUNT
                CHANGE_PASSWORD
        }
        session_id:   <SESSION_ID>         type: UUID
        # IF user_type = CHANGE_PASSWORD
        [
                new_password: <NEW_PASSWORD>
        ]
        # IF user_type = REGISTER
        [
                username: <USERNAME>
                password: <PASSWORD>
        ]
}

SERVER RESPONDS WITH FLOW-COMMONS/DATA TO CLIENT OF FORMAT
{
        status:       <STATUS>             type: String
        {
                OK
                # IF user_type = CLOSE_ACCOUNT
                [
                        OK
                        USERNAME_DOES_NOT_EXIST
                ]

                # IF user_type = REGISTER
                [
                        OK
                        USERNAME_TAKEN
                        USERNAME_INVALID
                        PASSWORD_INVALID
                ]

                # IF user_type = CHANGE_PASSWORD
                [
                        OK
                        PASSWORD_INVALID
                ]
        }
}
```

:: REQUESTING A LIST OF PROJECTS FOR A USER

Description: Client wants a list of projects they have access to.

```
CLIENT SENDS FLOW-COMMONS/DATA TO SERVER OF FORMAT
{
       type:          “list_projects”       type: String
}

SERVER RESPONDS WITH FLOW-COMMONS/DATA TO CLIENT OF FORMAT
{
       status:        <STATUS>              type: String
       {
              OK
       }
       # IF status = OK ELSE null
       [
              projects:     <PROJECTS>            type: UUID[]
       ]
}
```

:: REQUESTING A PROJECT
Description: Client wants a project.

```
CLIENT SENDS FLOW-COMMONS/DATA TO SERVER OF FORMAT
{
       type:          “project_info”        type: String
       project_uuid:  <PROJECT_UUID>        type: UUID
       session_id:    <SESSION_ID>          type: UUID
}

SERVER RESPONDS WITH FLOW-COMMONS/DATA TO CLIENT OF FORMAT
{
       status:        <STATUS>              type: String
       {
              OK
              INVALID_PROJECT_UUID
       }

       # IF status = OK ELSE null
       [
              project_name:         <PROJECT_NAME>        type: String
              owner:                <OWNER>               type: String
              editors:              <EDITORS>             type: String[]
              viewers:              <VIEWERS>             type: String[]
       ]
}
```

:: CREATE A NEW PROJECT
Description: Client wants to make a new project.

```
CLIENT SENDS DATA TO SERVER OF FORMAT
{
       type:          “new_project”         type: String
       project_name:  <PROJECT_NAME>        type: String
       session_id:    <SESSION_ID>          type: UUID
}

SERVER RESPONDS WITH DATA OF FORMAT
{
       status:        <STATUS>              type: String
       {
              OK
              PROJECT_NAME_INVALID
       }
```

```
        project_uuid:  <PROJECT>              type: UUID
}
```

:: MODIFYING A PROJECT
Description: Client wants to modify a project’s metadata.

```
CLIENT SENDS DATA TO SERVER OF FORMAT
{
        type:                 “project_modify”             type: String
        project_modify_type:  <PROJECT_MODIFY_TYPE>        type: String
        {
                MODIFY_COLLABORATOR
                RENAME_PROJECT
                DELETE_PROJECT
        }
        project_uuid:         <PROJECT_UUID>               type: UUID
        session_id:           <SESSION_ID>                 type: UUID

        # IF project_modify_type = MODIFY_COLLABORATOR
        [
                username:             <USERNAME>                   type: String
                access_level:         <ACCESS_LEVEL>               type: byte
                *Note: if the access level is ‘owner’ (3), then the previous owner will only have ‘edit’ (2)
access
                {
                        0 (NONE)
                        1 (VIEW)
                        2 (EDIT)
                        3 (OWNER)
                }
        ]s

        # IF project_modify_type = RENAME_PROJECT
        [
                new_name:             <NEW_NAME>                   type: String
        ]
}

SERVER RESPONDS WITH DATA OF FORMAT
{
        status:               <STATUS>                     type: String
        {
                OK
                INVALID_PROJECT_UUID

                # IF project_modify_type = MODIFY_COLLABORATOR
                [
                        USERNAME_DOES_NOT_EXIST
                        ACCESS_LEVEL_INVALID
                ]

                # IF project_modify_type = RENAME_PROJECT
                [
                        PROJECT_NAME_INVALID
                ]
        }
}
```

:: CREATE A NEW DIRECTORY
Description: Client wants to make a new directory.

```
CLIENT SENDS DATA TO SERVER OF FORMAT
```

```
{
       type:                        "new_directory"      type: String
       project_uuid:                <PROJECT_UUID>       type: UUID
       session_id:                  <SESSION_ID>         type: UUID
       parent_directory_uuid:       <PARENT_UUID>        type: UUID
       directory_name:              <DIRECTORY_NAME>     type: String
}

SERVER RESPONDS WITH DATA OF FORMAT
{
       status:               <STATUS>             type: String
       {
              OK
              DIRECTORY_NAME_INVALID
       }
       directory_uuid:              <UUID>        type: UUID
}
```

:: MODIFY A DIRECTORY

Description: Client wants to rename or delete a directory.

```
CLIENT SENDS DATA TO SERVER OF FORMAT
{
       type:                        "directory_modify"          type: String
       directory_uuid:              <DIRECTORY_UUID>            type: UUID
       session_id:                  <SESSION_ID>                type: UUID
       mod_type:                    <MOD_TYPE>                  type: String
       {
              RENAME
              DELETE
       }
       # IF mod_type = RENAME
       {
              new_name:             <NEW_NAME>                  type: String
       }
}

SERVER
{
       status:       <STATUS>             type: String
       {
              OK
              INVALID_DIRECTORY_UUID

              #IF mod_type = RENAME
              [
                     DIRECTORY_NAME_INVALID
              ]
       }
}
```

:: GET DIRECTORY INFO

```
CLIENT
{
       type:                 "directory_info"     type: String
       directory_uuid:       <DIRECTORY_UUID>     type: UUID
       session_id:           <SESSION_ID>         type: UUID
}
SERVER
{
       #IF directory_uuid != project_uuid
       [
               parent_directory_uuid:        <PARENT_DIRECTORY>    type: UUID
       ]
       directory_name:       <DIRECTORY_NAME>      type: String
       child_files:          <CHILD_FILES>         type: UUID[]
       child_directories:    <CHILD_DIRECTORIES>   type: UUID[]

       status:               <STATUS>              type: String
       {
               OK
               INVALID_DIRECTORY_UUID
       }
}
```

:: GET FILE INFO

```
Description: Client wants the metadata for a file
CLIENT
{
        type:                 “file_info”           type: String
        file_uuid:            <FILE_UUID>           type: UUID
        session_id:           <SESSION_ID>          type: UUID
}
SERVER
{
        file_name:            <FILE_NAME>           type: String
        file_versions:        <FILE_VERSIONS>       type: UUID[]
        file_type:            <FILE_TYPE>           type: String
        {
                TEXT_DOCUMENT
                ARBITRARY_DOCUMENT
        }

        status:               <STATUS>              type: String
        {
                OK
                INVALID_FILE_UUID
        }
}
```

:: REQUEST VERSION

```
Description: Client wants a specific version for version histories
CLIENT
{
        type:                 “version_request”     type: String
        file_uuid:            <FILE_UUID>           type: UUID
        version_uuid:         <VERSION_UUID>        type:UUID
        session_id:           <SESSION_ID>          type: UUID
}
SERVER
{
        version_data:         <VERSION_DATA>        type: byte[]
        status:               <STATUS>              type: String
        {
                OK
                INVALID_VERSION_UUID
        }
}
```

:: REQUEST VERSION INFO

```
Description: Client wants the date of the Version, given the UUID
CLIENT
{
        type:          “version_info”        type: String
        version_uuid:  <VERSION_UUID>        type: UUID
        session_id:    <SESSION_ID>          type: UUID
}
SERVER
{
        date:          <VERSION_SAVE_DATE>   type: Date
        status:        <STATUS>              type: String
        {
                OK
                INVALID_VERSION_UUID
        }
}
```

:: REQUEST LATEST VERSION OF A FLOWFILE
Description: Client wants the *Latest* Version of a file. Basically request_version with the latest UUID

```
CLIENT
{
        type:          "file_request"               type: String
        file_uuid:     <FILE_UUID>                  type: UUID
        session_id:    <SESSION_ID>                 type: UUID
}

SERVER
{
        file_data:     <FILE_DATA>                  type: byte[]
        version_uuid: <VERSION_UUID>                type: UUID
        status:        <STATUS>                     type: String
        {
                OK
                INVALID_FILE_UUID
                ACCESS_DENIED
        }
}
```

:: CREATE A NEW FLOWFILE (TEXT)
Description: Client wants to make a new text document.

```
CLIENT SENDS DATA TO SERVER OF FORMAT
{
        type:                 "new_text_file"       type: String
        file_name:            <FILE_NAME>           type: String
        session_id:           <SESSION_ID>          type: UUID
        directory_uuid:       <DIRECTORY_UUID>      type: UUID
}

SERVER RESPONDS WITH DATA OF FORMAT
{
        status:               <STATUS>              type: String
        {
                OK
                INVALID_DIRECTORY_UUID
                DOCUMENT_NAME_INVALID
        }
        file_uuid:            <FILE_UUID>           type: UUID
}
```

:: CREATE A NEW FLOWFILE (ARBITRARY)
Description: Client wants to create a new arbitrary FlowFile (e.g. database file, images, "not source code").

```
CLIENT SENDS DATA TO SERVER OF FORMAT
{
        type:                 "new_arbitrary_file"         type: String
        file_name:          <NEW_NAME>                  type: String
        file_data:          <FILE_DATA>                 type: byte[]
        session_id:         <SESSION_ID>                type: UUID
        directory_uuid:     <DIRECTORY_UUID>            type: UUID
}

SERVER
{
        status:             <STATUS>                    type: String
        {
                OK
                INVALID_DIRECTORY_UUID
                DOCUMENT_NAME_INVALID
        }
        file_uuid:          <FILE_UUID>                 type: UUID
}
```

:: MODIFY A FLOWFILE METADATA

Description: Client wants to rename or delete a FlowFile.

```
CLIENT SENDS DATA TO SERVER OF FORMAT
{
        type:               "file_metadata_modify"      type: String
        file_uuid:          <FLOWFILE_UUID>             type: UUID
        session_id:         <SESSION_ID>                type: UUID
        mod_type:           <MOD_TYPE>                  type: String
        {
                RENAME
                DELETE
        }
        # IF mod_type = RENAME
        {
                name:       <NEW_NAME>          type: String
        }
}

SERVER
{
        status:     <STATUS>            type: String
        {
                OK
                INVALID_DIRECTORY_UUID

                #IF mod_type = RENAME
                [
                        NAME_INVALID
                ]
        }
}
```

:: UPDATE A FLOWFILE (ARBITRARY)

Description: Client wants to update a non-source code file (e.g. images, database files).

```
CLIENT
{
        type:               "file_arbitrary_update"     type: String
        file_uuid:          <FILE_UUID>                 type: UUID
        project_uuid:       <PROJECT_UUID>              type: UUID
        session_id:         <SESSION_ID>                type: UUID
        mod_type:           <MOD_TYPE>                  type: String
        {
                UPDATE
                DELETE
        }

        # IF mod_type = UPDATE
        [
                new_data:      <NEW_DATA>                    type: byte[]
        ]
}

SERVER
{
        status:        <STATUS>              type: String
        {
                OK
                INVALID_FILE_UUID
                INVALID_DIRECTORY_UUID
        }
}
```

:: REGISTER LISTENER FOR FILE CHANGE

Description: Client wants to be updated from the server of changes to a file’s latest version

```
CLIENT
{
        type:          “file_async”          type: String
        session_id:    <SESSION_ID>          type: UUID
        rtype:         <REGISTER_TYPE>       type: String
        {
                REGISTER
                DEREGISTER
        }
        # IF rtype = REGISTER
        [
                file_uuid:            <FILE_UUID>           type: UUID
        ]
}
```

:: ASYNCHRONOUS CALLBACKS

:: MODIFY A FLOWFILE (TEXT)

Description: Server tells client that a FlowFile’s latest version has been modified.

```
CLIENT
{
        type:          “file_text_modify”          type: String
        file_uuid:     <FLOWFILE_UUID>             type: UUID
        session_id:    <SESSION_ID>                type: UUID

        mod_type:      <MOD_TYPE> (not HTML5)      type: String
        {
                INSERT
                DELETE
                MOVE
        }

        idx:           <IDX>                       type: Integer

        # IF mod_type = INSERT
        [
                str:   <STR>                       type: String
        ]
        #IF mod_type = DELETE
        [
                len:    <LEN>                        type: Integer
        ]
}

SERVER
{
        status:         <STATUS>            type: String
        {
                OK
                INDEX_OUT_OF_RANGE

                #IF doc_type = DELETE
                [
                        LENGTH_OUT_OF_RANGE
                ]
        }
}
```

User Interface

- Shared navbar:
  - shortcuts to editing, debug, settings
  - run, stop buttons (for both consoles)
- Login Perspective
  - Username
  - Password
  - New User
    - Username, password
- Editing Perspective
  - Document tree - list of projects and files in each project
    - Right click popup menus (new, delete, copy, cut, paste, share, rename, properties)
      - Properties: change the build path, etc.
  - Editing window
    - Syntax highlighting
    - Legend for the colours of each user’s cursor
    - Something to view images with
  - Console (errors and standard out)
  - “Editor” Toolbar
    - “Search”
    - “Share project”
    - “Export file”
    - “Import file”
- Debug Perspective
  - Variables list
  - Console
  - “Debug” Toolbar
    - Debug
    - Step in
    - Step over
    - Step out
- History Perspective
  - Vertical list of versions

    - Document tree
    - File viewer (no editing)
- Settings Perspective
    - Change user avatar (planned)
    - Change password
    - Log out
    - “Theme” - day/night (planned)
    - Close account
    - Set up Java environment

Edge Cases Considered

Client goes offline

- The client will be able to view code locally cached as “read-only”, no modifications will be permitted

Client disconnects while server is processing request

- IOException is thrown and captured, result of request is discarded

Error thrown while accessing database

- DatabaseException is thrown and captured, ‘INTERNAL_SERVER_ERROR’ message sent to client as the status of the result