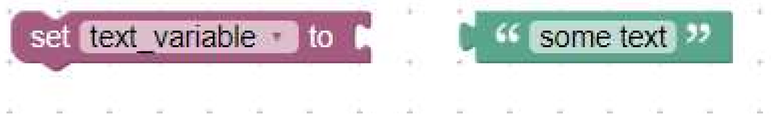
Basic Usage

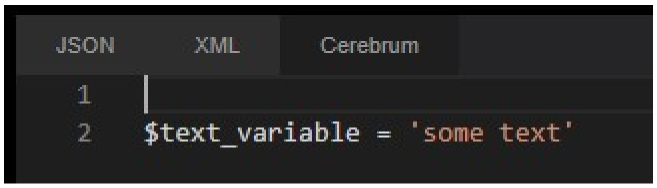
Drag blocks from the categories on the left hand side of the playground into the playground.
It should be visually obvious what kinds of blocks are eligible connections to each other

- The following is an example of assigning a variable to a text value. Note that we can use other data types as the variable's definition.
- To create a variable select the Variables category, click "Create Variable" and name the variable. In the same Variables category, we can now drag a variable set block into the workspace.



- We can connect these two blocks to set text_variable to "some text"

As we work in the Blockly workspace, with Cerebrum selected in the generators section, we will see Cerebrum code generated based on the blocks in our workspace. Below is the corresponding generator display when we connect the blocks shown above.

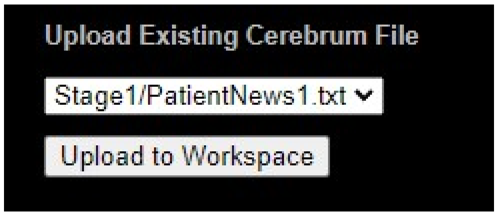


```

$text_variable = 'some text'
```

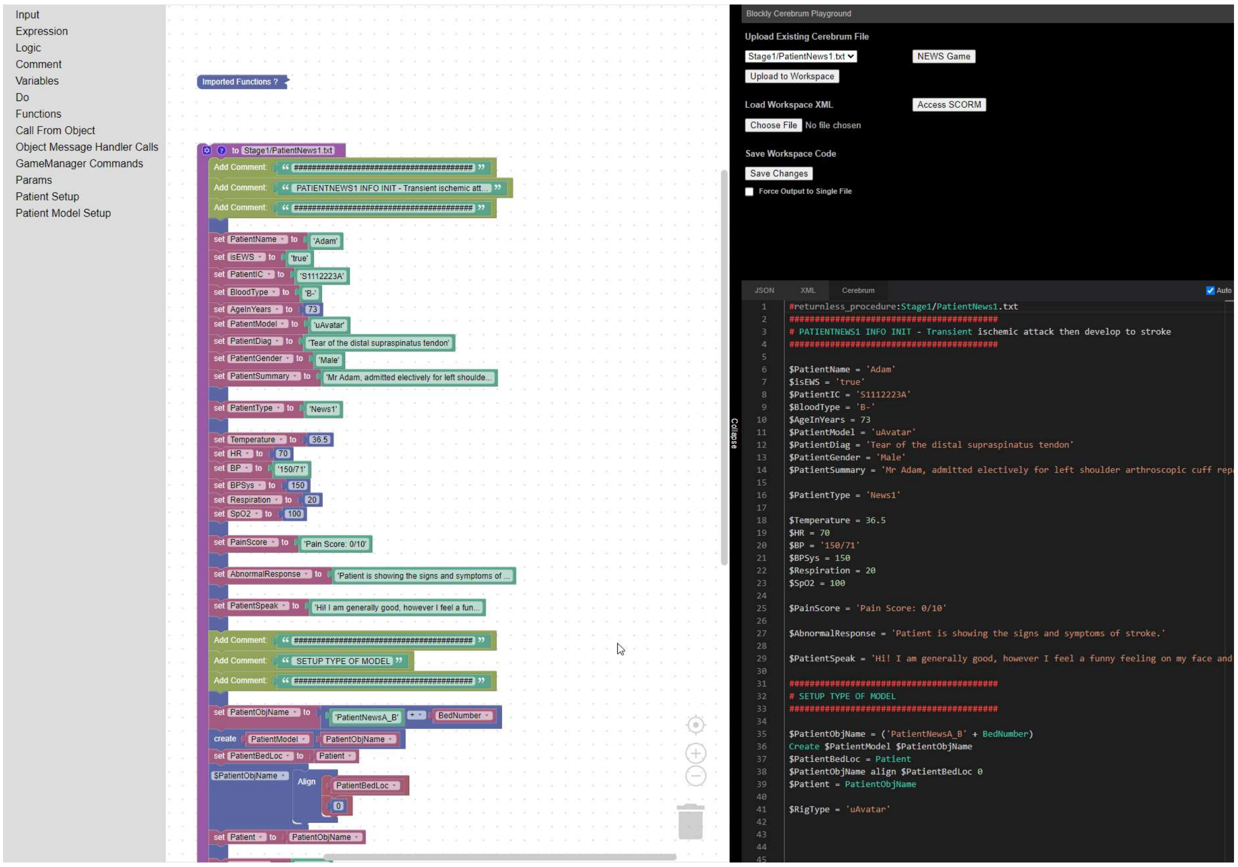
This is a single example of the general flow of using the editor and what we can expect from it as output.

Interacting With Existing Cerebrum Files

The Cerebrum Blockly Editor can be connected to an existing library of Cerebrum files. We can interact with these files and upload them to the workspace where we can make and save changes.



When we select a file from the dropdown and upload it to the workspace, we should see a large number of connected blocks that make up whatever file we've selected.



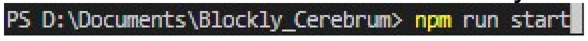
```
#returnless_procedure:Stage1/PatientNews1.txt
########################################
# PATIENTNEWS1 INFO INIT - Transient ischemic attack then develop to stroke
########################################

$PatientName = 'Adam'
$isEWS = 'true'
$PatientIC = 'S1112223A'
$BloodType = 'B-'
$AgeInYears = 73
$PatientModel = 'uAvatar'
$PatientDiag = 'Tear of the distal supraspinatus tendon'
$PatientGender = 'Male'
$PatientSummary = 'Mr Adam, admitted electively for left shoulder arthroscopic cuff repa

$PatientType = 'News1'

$Temperature = 36.5
$HR = 70
$BP = '150/71'
$BPSys = 150
$Respiration = 20
$SpO2 = 100

$PainScore = 'Pain Score: 0/10'

$AbnormalResponse = 'Patient is showing the signs and symptoms of stroke.'

$PatientSpeak = 'Hi! I am generally good, however I feel a funny feeling on my face and

########################################
# SETUP TYPE OF MODEL
########################################

$PatientObjName = ('PatientNewsA_B' + BedNumber)
Create $PatientModel $PatientObjName
$PatientBedLoc = Patient
$PatientObjName align $PatientBedLoc 0
$Patient = PatientObjName

$RigType = 'uAvatar'
```

We can see that the file we've selected has been imported to the workspace. Its contents are represented in blocks in the workspace and the corresponding Cerebrum code that is generated is shown on the right.

We can make changes to the values of any of the blocks that have been generated or add new ones and connect them. Our changes will be visible in the generator section. Once we're done making changes to the workspace, we can use the Save Changes button to save changes to the file. When we load it into the workspace again the file will include any saved changes we made prior.

<u>Setting Up a New Editor Implementation</u>

The Cerebrum Blockly Editor is really an extension of Google's native Blockly. Assuming we have the files for the Cerebrum Editor we can follow Blockly's instructions for setup - These are well documented and the process is the same for us as it is for a native Blockly user. We don't need to install Blockly with NPM since we already have the files. All we need to do is navigate to the root of the installation directory and enter:



```
PS D:\Documents\Blockly_Cerebrum> npm run start
```

This will start a node server which serves our Cerebrum Blockly implementation. By default the basic playground should open. The path we want to go to for the Cerebrum Editor is: "/tests/playgrounds/advanced_playground.html"

Creating and Handling New Block Types

The core of this process is again the same as it is for native Blockly (with some caveats). If we are creating a new block type AND our block type is NOT an Object Message Handler call block or a Game Manager call block, we should follow Blockly's documentation for adding Blocks to the library and adding handling for the generator for those blocks.

If we are creating an ObjectMessageHandler call block:

- Follow the instructions above and provided by Blockly's documentation. Add the definition for the block to Cerebrum/Blocks/ObjectMessageHandlerBlocks.js
- Add the generator code to Cerebrum/Generator/BindObjectMessageHandlers.js
- Add the new Block type's name as a string to the block registry array in Cerebrum/Helper/Builders/OMH_Calls.js

If we are creating a GameManager call block:

- Follow the instructions above and provided by Blockly's documentation. Add the definition for the block to Cerebrum/Blocks/GameManagerCallBlocks.js
- Add the generator code to Cerebrum/Generator/BindGameManagerCalls.js
- Add the new Block type's name as a string to the block registry array in Cerebrum/Helper/Builders/GameManager_RWORDS.js

In all other cases Blockly's default pattern of implementation should suffice for building new block types. We can put the definitions for new blocks in whichever of the existing files in Cerebrum/Blocks is most appropriate. If necessary we can create new files for new types of blocks.

- Each of these files contains a dictionary of block definitions that we need to export. We can import it in the CustomBlockLibrary.js file in Cerebrum/Blocks.

Likewise, we must add the generator definition for these new blocks in Cerebrum/Generator files. We can use whichever of the existing files is most appropriate.

- Each of these files contains a single function, bindBlockLib (named differently as is appropriate for the contents). This function must then be exported and imported in Cerebrum/Generator/CerebrumGenerator.mjs
- We must call this bindBlockLib function using the generator object in CerebrumGenerator.mjs.