File Sharing

File Sharing

- The real power of a data channel comes when combining it with other powerful technologies from a browser.
- By opening up the power to send data peer-to-peer and combining it with a File API, we could open up all new possibilities in your browser.
- This means you could add file sharing functionalities that are available to any user with an Internet connection.

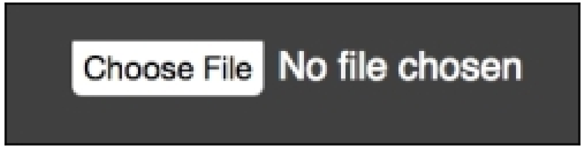
File Sharing

There will be a finite number of steps that both users will go through to transfer an entire file between them:

1. User A will open the page and type a unique ID.
2. User B will open the same page and type the same unique ID.
3. The two users can then connect to each other using RTCPeerConnection.
4. Once the connection is established, one user can select a file to share.
5. The other user will be notified of the file that is being shared, where it will be transferred to their computer over the connection and they will download the file.

Getting a file with the File API

- One of the first things that we will cover is how to use the File API to get a file from the user's computer.
- There is a good chance you have interacted with the File API on a web page and have not even realized it yet! The API is usually denoted by the Browse or Choose File text located on an input field in the HTML page and often looks something similar to this:



Getting a file with the File API

- Although there are many great features of the API, we are going to only focus on one small aspect of this API.
- This is the ability to get binary file data from the user by asking them to upload a file.
- A typical application that works with files, such as Notepad on Windows, will work with file data in pretty much the same way.
- It asks the user to open a file in which it will read the binary data from the file and display the characters on the screen.

Getting a file with the File API

- Now let's take a look at the following HTML code that demonstrates file sharing:

Refer to the file 7_1.txt

- The page should be fairly recognizable at this point. We will use the same page showing and hiding via CSS as done earlier.

Getting a file with the File API

- To start our code, we will build upon our application from the previous session.

- You can copy the files into a new folder for our file sharing application and add the HTML shown in the preceding section.

- You will also need all the steps from our JavaScript file to log in two users, create a WebRTC peer connection, and create a data channel between them.

Getting a file with the File API

- Copy the following code into your JavaScript file to get the page set up:

Refer to the file 7_2.txt

Getting a file with the File API

Getting a reference to a file

- Now that we have a simple page up and running, we can start working on the file sharing part of the application.
- The first thing the user needs to do is select a file from their computer's filesystem.
- This is easily taken care of already by the input element on the page.
- The browser will allow the user to select a file from their computer and then save a reference to that file in the browser for later use.

Getting a file with the File API

- When the user presses the Send button, we want to get a reference to the file that the user has selected.

- To do this, you need to add an event listener, as shown in the following code:

Refer to the file 7_3.txt

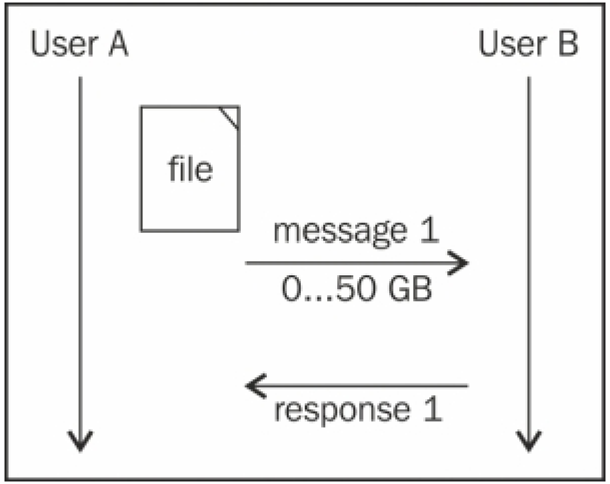
Getting a file with the File API

- Now, you might think that the object we get back will be in the form of the entire data inside of our file.
- What we actually get back from the input element is an object representing metadata about the file itself.
- Let's take a look at this metadata:

Refer to the file 7_4.txt

Breaking down a file into chunks

- Now that we have a file, we need to get it ready to send to the other user.
- The easiest approach to this problem is to take the entire file and call our send method with the data.
- This will take the entire file and give it to the data channel to process as one large message:



Breaking down a file into chunks

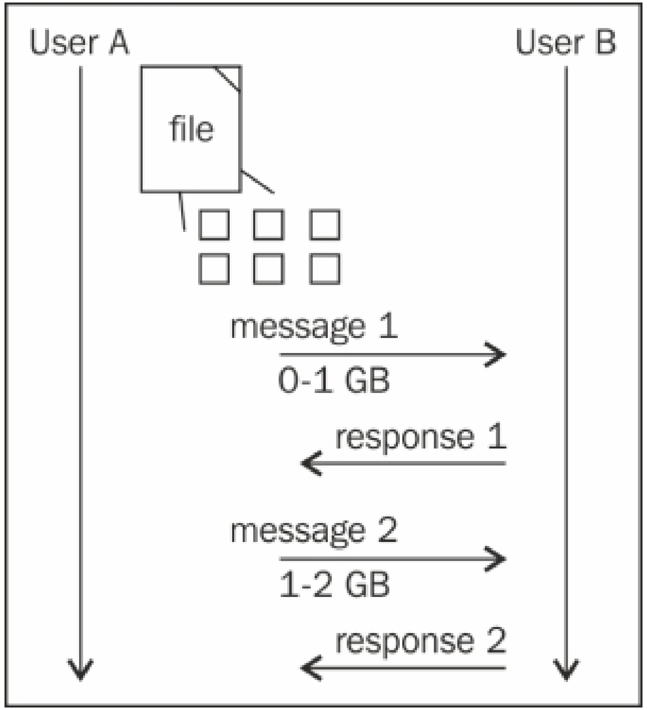
- We can apply this concept to our file just as easily. Consider a file to be one giant stream of 0s and 1s all in a row.

- This is known as our binary file data. The easiest way to chunk this file is to read a few lines of binary, send it over, then read the next few lines of binary and send this over.

- We can then keep doing this all the way until the end of the file.

Breaking down a file into chunks

- If we lose connection, we can just start over at the last chunk we sent, meaning the entire progress is not lost.



Breaking down a file into chunks

Making chunks readable

- Another topic that we have not touched on is exactly how to send binary data across the data channel.
- In our previous example, all of our data was already in a string format.
- This is due to the fact that it was in the form of simple text messages. This is a different story in the case of the File API, however.

Breaking down a file into chunks

- Turns out that life is never so easy! The unfortunate part is that there are many layers between JavaScript and the network protocol stack that handles the sending of data between clients.
- This stack speaks in an extremely specific and specialized way that requires you to translate the data into a more friendly and readable format.

Breaking down a file into chunks

- To get our code started, we will define a few simple methods that will encode and decode our data.

- You can define these anywhere in your script so that we can use them throughout the application that we are going to build:

Refer to the file 7_5.txt

Breaking down a file into chunks

- Now that we can encode data from one user, we need a function to decode data on the other side:

Refer to the file 7_6.txt

Breaking down a file into chunks

Reading and sending the file

- The next step is to actually read data from the file and send it to the other user.
- We will combine the data channel with the Base64 encoding to efficiently send the file.
- Let us implement the sendFile function that we used earlier in Getting a reference to a file subsection under Getting a file with the File API section in this session when getting a reference to the file:

Refer to the file 7_7.txt

Putting it together on the other side

- Now the other user should have a collection of chunks that make up the entire file.
- Since we used the ordered option in the data channel, the chunks should have come from the other user in order.
- This means we can put them together in one large data object as we receive them from the other side.

Putting it together on the other side

- We will also use the opposite decoding function that we covered in the Making chunks readable subsection under the Breaking down a file into chunks section in this session, to put our chunks into the right format:

Refer to the file 7_8.txt

Putting it together on the other side

- Our approach to saving the file is similar to the way the user would download a file from a website.
- Typically, a website will host a file on a server somewhere and provide a URL to the file. When the browser recognizes that the link is directly linked to a file, it will prompt the user on where they would like to save it or directly view it in the browser, depending on the type.

Putting it together on the other side

- Using a similar strategy, we can add the saveFile function to our project:

Refer to the file 7_9.txt

- The first thing our function does is take our file data and converts it to blob.

Showing progress to the user

- A final touch that we can add to improve our application is a progress meter to show how much of the file the user has and how much they have left.

- This makes for a better user experience so they can see when something is happening.

- We need to tell the first user how much data we have sent and the second user how much data has been received.

Showing progress to the user

- To show how much we have sent, we can add the following code to the sendChunk function:

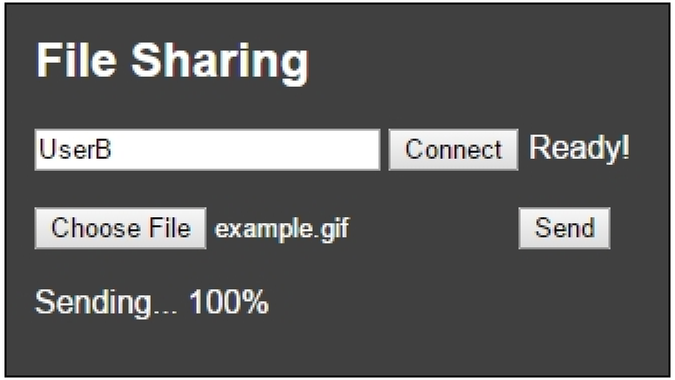
Refer to the file 7_10.txt

- We just take the current number of bytes sent versus the file's size and convert this into a percentage. Now, we can do the same for the receiving user:

Refer to the file 7_11.txt

Showing progress to the user

- We can apply the same logic on the other side, this time getting the current size from our last chunk's size.
- We then convert this to a percentage and show it to the user.
- You should now have a simple but effective file sharing application as shown here:



Sending... 100%

Summary

- It is pretty amazing what we were able to accomplish in just one small session.
- The example that we built works much like any file sharing application that is out there today.
- We were able to easily combine APIs together to create a fully featured experience in just a few hundred lines of code.
- It even works efficiently and accounts for several things, such as slow networks and large files.

Summary

- This is truly an example that you can expand upon and experiment with.
- There are plenty of ways to make this application much better to use.
- The first step could be printing out more information for the user to see, including what file they are sending and how large the file is.
- You can also give more information about the progress of the file and even the download speed.