```
package com.msoe.ce4960.sommere.lab5;

import java.net.InetAddress;
import java.net.UnknownHostException;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Fragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.text.Editable;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.EditText;

/**
 * Acts as a client to the SSFTP server over TCP
 *
 * Experiences:
 * This lab wasn't difficult at all as TCP is much easier to work with than UDP,
 * at least on the layer that Java uses it.
 *
 * Deficiencies:
 * There are two main issues with the application.  The IP address of the server
 * is hard coded in, and not all file types are supported.  Initially, it was
 * planned to add a server selector dialog, but it would require reworking too
 * much of the code to complete.  Android natively supports only a few select
 * file types, and those are handled when possible.
 *
 * Ideas for Improvement:
 * Overall the lab is pretty good, there isn't anything that I can think of
 * in need for improvement.
 *
 * Usage:
 * Update the ip address so it points to the server and then launch the application.
 * The usage is fairly self explanatory from there.
 *
 * Benchmarks:
 * To compare the performance of the UDP server to the TCP server, the following
 * benchmarks were run:
 *
 * | File Name            | File Size (Bytes) | UDP Avg. (us) | TCP Avg. (us) |
 * |----------------------|-------------------|---------------|---------------|
 * | test.txt             |                67 |         201.2 |         419.3 |
 * | Desert.jpg           |            845941 |   2,543,874.7 |     420,785.1 |
 * | 01 Couch Potato.mp3  |           4364416 |  13,116,291.1 |   2,316,369.2 |
 * |----------------------|-------------------|---------------|---------------|
 *
 * Each file was transfered 10 times on each protocol and the average was taken.
 * These numbers show that for transfers that only take a packet or two, UDP is
 * able to transfer the data in half the amount of time.  But for larger
```

```
 * transfers, such as images and music files, TCP transfers at a much faster
 * speed.  TCP is about 6 times faster than UDP for these transfers.  One
 * way to determine whether TCP or UDP should be used is to find out if the
 * overhead from TCP will be the majority of your packets.  If it is, then UDP
 * may be the better option.  But with TCP you get a whole host of reliability
 * and other features, which make it ideal for the larger transfers.
 *
 *
 * @author Erik Sommer
 */

/**
 * Main activity of the application.  Handles the initialization
 * @author Erik Sommer
 *
 */
public class MainActivity extends Activity{


    /**
     * Manages the list of files
     */
    private FileIndexFragment mFileIndexFragment;

    /**
     * Address of the server
     */
    private InetAddress mServerAddress;

    /**
     * Handles when a fragment attaches to the activity
     * @param fragment  the fragment that was attached
     */
    @Override
    public void onAttachFragment(Fragment fragment) {

        super.onAttachFragment(fragment);

        // Check if the fragment is an instance of one it is looking for
        if(fragment instanceof FileIndexFragment){

            // If so, Store a reference
            mFileIndexFragment = (FileIndexFragment) fragment;

            // Set the IP for the fragment to use
            mFileIndexFragment.setIP(mServerAddress);

        }
    }

    /**
     * Executed when the activity is started
     * @param savedInstanceState    state to restore, if any
```

```
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        // Get the server address
        try {
            mServerAddress = InetAddress.getByName("192.168.1.8");
        } catch (UnknownHostException e) {
            // Log an error if the server can't be found and stop the activity
            Log.e(getClass().getName(), "Unable to find server.", e);
            finish();
        }

        // Set the XML view for the program
        setContentView(R.layout.main);
    }

    /**
     * Handles when the options menu needs to be created
     * @param   menu    the menu to inflate the layout into
     */
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        // Inflate the layout
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main_activity, menu);

        return true;
    }

    /**
     * Handles when a menu item is selected
     * @param item   the menu item is selected
     */
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {

        // Check if it's the refresh button
        if(item.getItemId() == R.id.menu_refresh){

            // If it is, trigger a refresh
            if(mFileIndexFragment != null){
                mFileIndexFragment.refresh();
            }

            return true;

        }else if(item.getItemId() == R.id.menu_edit){

            // Otherwise, if it's the edit button
```

```
            final EditText input = new EditText(this);
            input.setText(mServerAddress.getHostAddress());

            // Build a server selector dialog
            new AlertDialog.Builder(this)
            .setTitle("Server IP Address")
            .setMessage("Enter the Server IP Address")
            .setView(input)
            .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    Editable value = input.getText();
                    try {

                        // Validate the user input and set the server
                        if(value.toString().length() == 0){
                            return;
                        }

                        mServerAddress = InetAddress.getByName(value.toString());
                        mFileIndexFragment.setIP(mServerAddress);
                        mFileIndexFragment.refresh();
                    } catch (UnknownHostException e) {
                        Log.e(getClass().getName(), "Unable to find server", e);
                    }
                }
            }).setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int whichButton) {
                    // Do nothing.
                }
            }).show();
            return true;
        }else{
            return super.onOptionsItemSelected(item);
        }
    }
}
```

```java
package com.msoe.ce4960.sommere.lab5;


/**
 * Represents a stateless file server packet
 * @author Erik Sommer
 *
 */
public class SSFTP {

    /**
     * File name used to indicate a directory listing request/response
     */
    public static final String DIR_LISTING_FILE = ".";

    /**
     * Number of bytes the file name may take up
     */
    public static final int FILE_NAME_NUM_BYTES = 32;

    /**
     * Bit mask for the EOF flag
     */
    private static final int IS_EOF_MASK = (1 << 1);

    /**
     * Bit mask for the file not found flag
     */
    private static final int IS_FILE_NOT_FOUND_MASK = (1 << 2);

    /**
     * Bit mask for the invalid request flag
     */
    private static final int IS_INVALID_REQUEST_MASK = (1 << 3);

    /**
     * Bit mask for the request flag
     */
    private static final int IS_REQUEST_MASK = (1 << 0);

    /**
     * The maximum length of the the file name.  Minus 1 is necessary for the
     * null terminator
     */
    private static final int MAX_FILE_NAME_LENGTH = FILE_NAME_NUM_BYTES - 1;

    /**
     * Number of bytes in the header
     */
    public static final int NUM_HEADER_BYTES = 39;

    /**
     * Maximum size of the data that can be returned
```

```
     */
    private static final int MAX_LENGTH = 14000;

    /**
     * Maximum offset of the data that can be returned
     */
    private static final long MAX_OFFSET = (int)Math.pow(2, 32);

    /**
     * Creates an {@link SSFTP} object from a stream of bytes
     * @param bytes the stream of bytes to convert
     * @return      an SSFTP object based off of the stream of the bytes
     */
    public static SSFTP fromBytes(byte[] bytes){

        int buffPos = 0;

        // Grab the flags
        byte flags = bytes[buffPos++];

        // Grab  the length
        int length = (unsignedByteToInt(bytes[buffPos++]) << 8)
                + unsignedByteToInt(bytes[buffPos++]);

        // Grab the offset
        long offset = (unsignedByteToInt(bytes[buffPos++]) << 24)
                + (unsignedByteToInt(bytes[buffPos++]) << 16)
                + (unsignedByteToInt(bytes[buffPos++]) << 8)
                + unsignedByteToInt(bytes[buffPos++]);

        // Grab the start of the name and find the end of it
        int nameStartIndex = buffPos;

        while(bytes[buffPos] != 0){
            buffPos++;
        }

        // Calculate the length of the name and extract it
        int nameLength = buffPos - nameStartIndex;

        byte[] nameBytes = new byte[nameLength];
        System.arraycopy(bytes, nameStartIndex, nameBytes, 0, nameLength);

        // Validate the file name
        String fileName = new String(nameBytes);

        boolean nameInvalid = (fileName.length() == 0);

        if(nameInvalid){
            fileName = " ";
        }

        // Validate the size
```

```
        boolean sizeInvalid = ((length > MAX_LENGTH) || (length < 0));

        if(sizeInvalid){
            length = 0;
        }

        // Create the new SSFTP object
        SSFTP ssftp = new SSFTP(fileName, length, offset);

        // Set the flags
        ssftp.mIsRequest = ((flags & IS_REQUEST_MASK) == IS_REQUEST_MASK);
        ssftp.mIsEOF = ((flags & IS_EOF_MASK) == IS_EOF_MASK);
        ssftp.mIsFileNotFound = ((flags & IS_FILE_NOT_FOUND_MASK) == IS_FILE_NOT_FOUND_MASK);
        ssftp.mIsInvalidRequest = ((flags & IS_INVALID_REQUEST_MASK) == IS_INVALID_REQUEST_MASK);

        // Set a flag if the file name is not valid
        if((nameInvalid) || sizeInvalid){
            ssftp.setIsInvalidRequest(true);
        }

        // Move the pointer to the data
        buffPos += MAX_FILE_NAME_LENGTH - nameLength + 1;

        return ssftp;
    }

    /**
     * Converts an unsigned byte to a java {@code int}
     * @param b the byte to convert
     * @return  the {@code int} form of the byte
     */
    private static int unsignedByteToInt(byte b){
        return (int) b & 0xFF;
    }

    /**
     * Name of the file
     */
    private String mFileName;

    /**
     * Indicates whether the EOF has been reached
     */
    private boolean mIsEOF;

    /**
     * Indicates whether the file was not found
     */
    private boolean mIsFileNotFound;

    /**
     * Indicates whether the request is invalid
     */
```

```
    private boolean mIsInvalidRequest;

    /**
     * Indicates whether this packet is a request or response
     */
    private boolean mIsRequest;

    /**
     * Number of bytes requested/returned
     */
    private int mLength;

    /**
     * Offset to start reading at
     */
    private long mOffset;

    /**
     * Size of the data
     */
    private int mDataSize;

    /**
     * Constructor.  Creates a new request packet
     * @param fileName  name of the file
     * @param length    number of bytes to request
     * @param offset    start position to start reading
     */
    public SSFTP(String fileName, int length, long offset){

        // Argument Validation
        // TODO: Fix the unsigned range issue (only have half the range)
        if(fileName == null){
            throw new IllegalArgumentException("fileName is null");
        }else if((fileName.length() == 0) || (fileName.length() > MAX_FILE_NAME_LENGTH)){
            throw new IllegalArgumentException("fileName is not between 1 and " +
            MAX_FILE_NAME_LENGTH + " characters");
        }else if((length > MAX_LENGTH) || (length < 0)){
            throw new IllegalArgumentException("length is not between 0 and " + MAX_LENGTH);
        }else if((offset < 0) || (offset > MAX_OFFSET)){
            throw new IllegalArgumentException("offset is not between 0 and " + MAX_OFFSET);
        }

        mLength = length;
        mFileName = fileName;
        mOffset = offset;

        // Set the flags
        mIsRequest = true;
        mIsEOF = false;
        mIsFileNotFound = false;
        mIsInvalidRequest = false;
    }
```

```
    /**
     * Gets the file name
     * @return  the file name
     */
    public String getFileName(){
        return this.mFileName;
    }

    /**
     * Gets the length of the data requested/returned
     * @return  the length of the data requested/returned
     */
    public int getLength(){
        return this.mLength;
    }

    /**
     * Gets the size of the packet (including the header and data)
     * @return  the size of the packet
     */
    public int getNumBytes(){
        return NUM_HEADER_BYTES + mDataSize;
    }

    /**
     * Gets the offset
     * @return  the offset
     */
    public long getOffset(){
        return this.mOffset;
    }

    /**
     * Indicates whether the EOF flag has been set
     * @return  {@code true} if EOF flag has been set, {@code false} otherwise
     */
    public boolean isEOF(){
        return mIsEOF;
    }

    /**
     * Indicates whether the FileNotFound flag has been set
     * @return  {@code true} if the FileNotFound flag has been set,
     *          {@code false} otherwise
     */
    public boolean isFileNotFound(){
        return this.mIsFileNotFound;
    }

    /**
     * Indicates whether the InvalidRequest flag has been set
     * @return  {@code true} if the InvalidRequest flag has been set,
```

```
     *          {@code false} otherwise
     */
    public boolean isInvalidRequest(){
        return this.mIsInvalidRequest;
    }

    /**
     * Indicates whether this packet is a request or response
     * @return  {@code true} if this packet is a request,
     *          {@code false} if this packet is a response
     */
    public boolean isRequest(){
        return mIsRequest;
    }

    /**
     * Gets whether this packet is a response
     * @return  {@code true} if this packet is a response, {@code false} if
     *          this packet is a response
     */
    public boolean isResponse(){
        return !mIsRequest;
    }

    /**
     * Sets the file name
     * @param fileName  the file name
     * @throws IllegalArgumentException if {@code fileName} is null, has a
     *                                  length of 0, or is greater than
     *                                  {@code MAX_FILE_NAME_LENGTH} characters
     */
    public void setFileName(String fileName){

        if(fileName == null){
            throw new IllegalArgumentException("fileName is null");
        }else if((fileName.length() == 0) || (fileName.length() > MAX_FILE_NAME_LENGTH)){
            throw new IllegalArgumentException("fileName is not between 1 and " +
            MAX_FILE_NAME_LENGTH + " characters");
        }

        this.mFileName = fileName;
    }

    /**
     * Sets whether the EOF flag has been set
     * @param isEOF {@code true} if the EOF flag has been set, {@code false}
     *              otherwise
     */
    public void setIsEOF(boolean isEOF){
        this.mIsEOF = isEOF;
    }

    /**
```

```
     * Sets whether the FileNotFound flag has been set
     * @param isFileNotFound    {@code true} if the FileNotFound flag has been
     *                          set, {@code false} otherwise
     */
    public void setIsFileNotFound(boolean isFileNotFound){
        this.mIsFileNotFound = isFileNotFound;
    }

    /**
     * Sets whether the InvalidRequest flag has been set
     * @param isInvalidRequest  {@code true} if the InvalidRequest flag has been
     *                          set, {@code false} otherwise
     */
    public void setIsInvalidRequest(boolean isInvalidRequest){
        this.mIsInvalidRequest = isInvalidRequest;
    }

    /**
     * Sets whether the packet is a request or a response
     * @param isRequest {@code true} if this packet is a request, {@code false}
     *                  if this packet is a response
     */
    public void setIsRequest(boolean isRequest){
        this.mIsRequest = isRequest;
    }

    /**
     * Sets whether this packet is a response
     * @param isResponse    {@code true} if this packet is a response,
     *                      {@code false} if this packet is a request
     */
    public void setIsResponse(boolean isResponse){
        mIsRequest = !isResponse;
    }

    /**
     * Sets the length of the data requested/returned
     * @param length    the length of the data requested/returned
     * @throws IllegalArgumentException if length is less than 0 or greater than {@link
     MAX_LENGTH}
     */
    public void setLength(int length){

        if((length > MAX_LENGTH) || (length < 0)){
            throw new IllegalArgumentException("length is not between 0 and " + MAX_LENGTH);
        }

        this.mLength = length;
    }

    public void setOffset(long offset){

        if((offset < 0) || (offset > MAX_OFFSET)){
```

```
            throw new IllegalArgumentException("offset is not between 0 and " + MAX_OFFSET);
        }
    }

    /**
     * Converts the {@link SSFTP} object into bytes
     * @return  an array of bytes that represent the SSFTP object
     */
    public byte[] toBytes(){

        // Create the array (size of the header + size of the data)
        byte returnBytes[] = new byte[NUM_HEADER_BYTES + mDataSize];
        int buffPos = 0;

        // Set the flags
        int flags = 0;

        if(mIsRequest){
            flags |= IS_REQUEST_MASK;
        }

        if(mIsEOF){
            flags |= IS_EOF_MASK;
        }

        if(mIsFileNotFound){
            flags |= IS_FILE_NOT_FOUND_MASK;
        }

        if(mIsInvalidRequest){
            flags |= IS_INVALID_REQUEST_MASK;
        }

        returnBytes[buffPos++] = (byte)(0xFF & flags);

        // Set the length
        returnBytes[buffPos++] = (byte)((0xFF00 & mLength) >> 8);
        returnBytes[buffPos++] = (byte)(0x00FF & mLength);

        returnBytes[buffPos++] = (byte)((0xFF000000 & mOffset) >> 24);
        returnBytes[buffPos++] = (byte)((0x00FF0000 & mOffset) >> 16);
        returnBytes[buffPos++] = (byte)((0x0000FF00 & mOffset) >> 8);
        returnBytes[buffPos++] = (byte)(0x000000FF & mOffset);

        // Set the file name
        byte[] fileNameBytes = mFileName.getBytes();

        System.arraycopy(fileNameBytes, 0, returnBytes, buffPos, fileNameBytes.length);

        buffPos += fileNameBytes.length;

        // Fill the remaining area with zeroes (null termination)
        for(int i = fileNameBytes.length; i < FILE_NAME_NUM_BYTES; i++){
```

```
            returnBytes[buffPos++] = 0;
        }

        return returnBytes;
    }

    /**
     * Prints out the string representation of the packet
     */
    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();

        builder.append("Read Back\nIsRequest:\t\t" + String.valueOf(mIsRequest));
        builder.append("\nIsEOF:\t\t\t" + String.valueOf(mIsEOF));
        builder.append("\nIsFileNotFound:\t\t" + String.valueOf(mIsFileNotFound));
        builder.append("\nIsInvalidRequest:\t" + String.valueOf(mIsInvalidRequest));
        builder.append("\nLength:\t\t\t" + String.valueOf(mLength));
        builder.append("\nOffset:\t\t\t" + String.valueOf(mOffset));
        builder.append("\nFileName:\t\t\"" + String.valueOf(mFileName) + "\"");

        return builder.toString();
    }
}
```

```java
package com.msoe.ce4960.sommere.lab5;

import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;

import android.os.AsyncTask;
import android.util.Log;

/**
 * AsyncTask that fetches the file from the server
 * @author Erik Sommer
 *
 */
public class FileFetcher extends AsyncTask<String, Void, byte[]> {

    /**
     * Name of the file that is downloaded
     */
    private String mFileName;

    /**
     * Interface used to handle when the file has been fetched
     * @author Erik Sommer
     *
     */
    public interface FileFetcherListener{

        /**
         * Handles when the file has been fetched
         * @param fileName  the name of the file
         * @param data      the data contained in the file
         */
        public void onFileFetched(String fileName, byte[] data);

    }

    /**
     * Address of the server
     */
    private InetAddress serverAddress;

    /**
     * Listener to call back to when the data has been downloaded
     */
    private FileFetcherListener mListener;
```

```
    /**
     * Constructor
     * @param listener      listener to call back to when the data has been fetched
     * @param serverAddress server to get the data from
     */
    public FileFetcher(FileFetcherListener listener, InetAddress serverAddress){
        mListener = listener;
        this.serverAddress = serverAddress;
    }

    /**
     * Port to connect to on the server
     */
    private static final int SERVER_PORT = 22222;

    /**
     * Size of the request
     */
    private static final int REQUEST_SIZE = 5000;

    /**
     * Fetches the file
     * @param params    first parameter should be the file name.  All others
     *                  are ignored
     * @return  a {@code byte[]} array containing the file contents
     */
    @Override
    protected byte[] doInBackground(String... params) {

        // Grab the file name and create a new packet
        String fileName = params[0];
        SSFTP ssftp = new SSFTP(fileName, REQUEST_SIZE, 0);


        Socket clientSocket;
        byte data[] = null;

        try{
            // Write out the request
            clientSocket = new Socket(serverAddress, SERVER_PORT);
            DataOutputStream outputStream = new DataOutputStream(clientSocket.getOutputStream());
            outputStream.write(ssftp.toBytes());

            // Read in the response
            byte[] responsePacket = new byte[SSFTP.NUM_HEADER_BYTES];
            DataInputStream inputStream = new DataInputStream(clientSocket.getInputStream());
            inputStream.read(responsePacket);

            // Create the SSFTP object and validate it
            SSFTP responsePakcet = SSFTP.fromBytes(responsePacket);

            // Validate the response
            if(responsePakcet.isInvalidRequest()){
```

```
                Log.w(getClass().getName(), "Request is invalid");
                return null;
            }else if(responsePakcet.isFileNotFound()){
                Log.w(getClass().getName(), "File was not found");
                return null;
            }

            // The response is as expected, start reading the file
            DataInputStream networkInput = new DataInputStream(clientSocket.getInputStream());
            ByteArrayOutputStream out = new ByteArrayOutputStream();


            byte[] buffer = new byte[REQUEST_SIZE];
            int numRead = 0;

            // Read in the file data
            while((numRead = networkInput.read(buffer)) != -1){
                out.write(buffer, 0, numRead);
            }

            // Convert the stream to an array
            data = out.toByteArray();

            // Close the streams
            try {
                inputStream.close();
            }catch (IOException e){
                Log.e(getClass().getName(), "Error closing the input stream", e);
            }

            try {
                networkInput.close();
            }catch (IOException e){
                Log.e(getClass().getName(), "Error closing the input stream", e);
            }

            try {
                clientSocket.close();
            }catch (IOException e){
                Log.e(getClass().getName(), "Error closing the input stream", e);
            }
        } catch (SocketException e) {
            Log.e(getClass().getName(), "A socket error occured", e);
        } catch (UnknownHostException e) {
            Log.e(getClass().getName(), "Could not find server", e);
        } catch (IOException e) {
            Log.e(getClass().getName(), "An I/O error occured", e);
        }

        mFileName = fileName;

        return data;
    }
```

```
    /**
     * Handles when the file has finished downloading.  Notifies the
     * listener
     * @param result    the file that was read
     */
    @Override
    protected void onPostExecute(byte[] result) {
        mListener.onFileFetched(mFileName, result);
    }
}
```

```java
package com.msoe.ce4960.sommere.lab5;

import java.net.InetAddress;

import android.app.ListFragment;
import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import com.msoe.ce4960.sommere.lab5.FileFetcher.FileFetcherListener;

/**
 * Fragment that creates and manages the index of files that are available from
 * the server.
 * @author Erik Sommer
 *
 */
public class FileIndexFragment extends ListFragment implements FileFetcherListener {

    /**
     * Address of the server
     */
    private InetAddress serverAddress;

    /**
     * Handles when the activity is created.  Sets up the {@link ListView} and
     * fetches the index
     * @param   savedInstanceState  the state to restore
     */
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);

        // Set so only one item can be chosen
        getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);

        // Request the index
        new FileFetcher(this, serverAddress).execute(".");
    }

    /**
     * Handles when the view is created
     * @param inflater              the inflater to use to inflate the view
     * @param container             the container for the view
     * @param savedInstanceState    the state to restore
     * @return  the view to display
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
```

```
            Bundle savedInstanceState) {

        // Inflate the view and return
        return inflater.inflate(R.layout.file_index_fragment, null);
    }

    /**
     * Handles when the index has been fetched
     * @param fileName  the name of the file
     * @param data      the data that was received
     */
    public void onFileFetched(String fileName, byte[] data) {

        // Create a new string with the data
        String dataString = new String(data);

        // Split the string based on the delimiters
        String[] files = dataString.split("\\r?\\n");

        // Create a new adapter for the list
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(getActivity(),
                android.R.layout.simple_list_item_activated_1, files);

        // Update the adapter
        setListAdapter(adapter);
    }

    /**
     * Handles when an item has been clicked
     * @param l         the {@link ListView} that contains the item that has been
     *                  clicked
     * @param v         the view that was clicked
     * @param position  the position of the view that was clicked
     * @param id        the id of the view that was clicked
     */
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {

        // Get a new intent for the viewer
        Intent intent = new Intent(getActivity(), ViewerActivity.class);

        // Add the file name and the server IP
        intent.putExtra(ViewerActivity.KEY_FILE_NAME,
                (String)getListAdapter().getItem(position));
        intent.putExtra(ViewerActivity.KEY_SERVER_IP,
                serverAddress.getHostAddress());

        // Start the viewer
        startActivity(intent);
    }

    /**
     * Triggers a refresh of the server index
```

```
     */
    public void refresh(){
        new FileFetcher(this, serverAddress).execute(".");
    }

    /**
     * Sets the server address
     * @param serverAddress the address of the server
     */
    public void setIP(InetAddress serverAddress) {
        this.serverAddress = serverAddress;
    }
}
```

```java
package com.msoe.ce4960.sommere.lab5;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileDescriptor;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.UnknownHostException;

import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.content.Intent;
import android.media.MediaPlayer;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.widget.Toast;

import com.msoe.ce4960.sommere.lab5.FileFetcher.FileFetcherListener;

/**
 * Handles the display of the downloaded file
 * @author Erik Sommer
 *
 */
public class ViewerActivity extends Activity implements FileFetcherListener {

    /**
     * Name of the file that was displayed
     */
    private String mFileName;

    /**
     * Initializes the activity
     * @param savedInstanceState    state to restore
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Get the file name
        mFileName =  getIntent().getStringExtra(KEY_FILE_NAME);

        // Notify the user the file is being fetched
        Toast.makeText(this, "Fetching: " + mFileName, Toast.LENGTH_LONG).show();

        // Set the view
```

```
        setContentView(R.layout.viewer_activity);

        // Start fetching the file
        try {
            new FileFetcher(this, InetAddress.getByName(getIntent().getStringExtra("ipAddress"
            ))).execute(mFileName);
        } catch (UnknownHostException e) {
            Log.e(getClass().getName(), "Could not find server", e);
        }
    }

    /**
     * Key for the name of the file
     */
    public static final String KEY_FILE_NAME = "fileName";

    /**
     * Key for the ip address of the server
     */
    public static final String KEY_SERVER_IP = "ipAddress";

    /**
     * Handles when the file has been fetched
     * @param fileName  the name of the file that has been fetched
     * @param data      the data of the file
     */
    public void onFileFetched(String fileName, byte[] data) {

        boolean mExternalStorageWriteable = false;
        String state = Environment.getExternalStorageState();

        // Determine whether the image can be written
        if (Environment.MEDIA_MOUNTED.equals(state)) {
            // Data can be read and written
            mExternalStorageWriteable = true;
        } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
            // Data can only be read
            mExternalStorageWriteable = false;
        } else {
            // Data can't be read or written
            mExternalStorageWriteable = false;
        }

        File file = null;

        if(mExternalStorageWriteable){
            // Create the file
            file = new File(getExternalFilesDir(null), fileName);

            try {
                // Convert the byte array to a stream and write the file
                ByteArrayInputStream is = new ByteArrayInputStream(data);
                OutputStream os = new FileOutputStream(file);
```

```
            is.read(data);
            os.write(data);
            is.close();
            os.close();
        } catch (IOException e) {
            Log.w(getClass().getName(), "I/O error occured", e);
        }
    }

    // Find the extension of the file
    String filenameArray[] = fileName.split("\\.");
    String extension = filenameArray[filenameArray.length-1];

    if(extension.equalsIgnoreCase("txt")){
        // Use a text viewer if it is text
        TextFragment textFragment  = new TextFragment(file);
        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
        fragmentTransaction.add(R.id.viewer_container, textFragment);
        fragmentTransaction.commit();
    }else if(extension.equalsIgnoreCase("jpg")){
        // Use an image view if it is a jpg
        ImageFragment imageFragment  = new ImageFragment(file);
        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
        fragmentTransaction.add(R.id.viewer_container, imageFragment);
        fragmentTransaction.commit();
    }else if(extension.equalsIgnoreCase("mp3")){
        // Play back the file if it is an MP3

            FileDescriptor fd = null;

            try {
                // Get the stream
                FileInputStream fis = new FileInputStream(file);
                fd = fis.getFD();


                if (fd != null) {
                    // Start playback
                    MediaPlayer mediaPlayer = new MediaPlayer();
                    mediaPlayer.setDataSource(fd);
                    mediaPlayer.prepare();
                    mediaPlayer.start();
                }

            } catch (FileNotFoundException e) {
                Log.e(getClass().getName(), "File not found", e);
            } catch (IOException e) {
                Log.e(getClass().getName(), "I/O exception occured", e);
            }
    }else{
        // Otherwise, try starting an intent with the file, maybe an
```

```
            // application on the device can handle it
            Intent intent = new Intent();
            intent.setAction(android.content.Intent.ACTION_VIEW);
            intent.setData(Uri.fromFile(file));
            startActivity(intent);
            finish();
            Toast.makeText(getApplicationContext(),
                    "Unsupported file type: \"" + extension + "\"", Toast.LENGTH_SHORT);
        }
    }
}
```

```java
package com.msoe.ce4960.sommere.lab5;

import java.io.File;

import android.app.Fragment;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.os.Environment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;

/**
 * Display an image that has been received
 * @author Erik Sommer
 *
 */
public class ImageFragment extends Fragment {

    /**
     * Image file to display
     */
    private File mFile;

    /**
     * Constructor.
     * @param file  the file to display
     */
    public ImageFragment(File file){
        mFile = file;

    }

    /**
     * Creates the view with the image
     * @param inflater              the inflater used to inflate the view
     * @param container             the container for the view
     * @param savedInstanceState    the state to restore
     * @return  the view to display
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {

        // Create the view
        View view = inflater.inflate(R.layout.image_fragment, null);

        boolean mExternalStorageAvailable = false;
        String state = Environment.getExternalStorageState();

        // Determine whether the image can be read
```

```
        if (Environment.MEDIA_MOUNTED.equals(state)) {
            // Data can be read and written
            mExternalStorageAvailable = true;
        } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
            // Data can only be read
            mExternalStorageAvailable = true;
        } else {
            // Data can't be read or written
            mExternalStorageAvailable = false;
        }

        // If getting the file is possible
        if(mExternalStorageAvailable){

            // Get and decode the file
            File file = mFile;
            Bitmap myBitMap = BitmapFactory.decodeFile(file.getAbsolutePath());

            // Display the file
            ((ImageView)view.findViewById(R.id.imageview_fragment)).setImageBitmap(myBitMap);
        }

        return view;
    }


}
```

```java
package com.msoe.ce4960.sommere.lab5;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import android.app.Fragment;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

/**
 * Handles the display of text data
 * @author Erik Sommer
 *
 */
public class TextFragment extends Fragment {

    /**
     * File to read the text from
     */
    private File mFile;

    /**
     * Constructor.
     * @param file  the file to display
     */
    public TextFragment(File file){
        mFile = file;

    }

    /**
     * Creates the view with the image
     * @param inflater              the inflater used to inflate the view
     * @param container             the container for the view
     * @param savedInstanceState    the state to restore
     * @return  the view to display
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {

        // Create the view
        View view = inflater.inflate(R.layout.text_fragment, null);

        boolean mExternalStorageAvailable = false;
```

```
        String state = Environment.getExternalStorageState();

        // Determine whether the image can be read
        if (Environment.MEDIA_MOUNTED.equals(state)) {
            // Data can be read and written
            mExternalStorageAvailable = true;
        } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
            // Data can only be read
            mExternalStorageAvailable = true;
        } else {
            // Data can't be read or written
            mExternalStorageAvailable = false;
        }

        // If getting the file is possible
        if(mExternalStorageAvailable){

            File file = mFile;
            BufferedReader reader;

            try {
                // Create a reader for the file
                reader = new BufferedReader(new FileReader(file));

                StringBuilder builder = new StringBuilder();
                char buffer[] = new char[1000];

                // Read in the file
                while(reader.read(buffer) != -1){
                    builder.append(buffer);
                }

                // Set the text
                ((TextView)view.findViewById(R.id.text_fragment))
                .setText(builder.toString());

            } catch (FileNotFoundException e) {
                Log.e(getClass().getName(), "File could not be found", e);
            } catch (IOException e) {
                Log.e(getClass().getName(), "I/O error reading the file", e);
            }
        }

        return view;
    }


}
```