Introduction

Overview

This document shows how to implement a FileSelector control for Android.
This document contains information on how to implement the control using the Android SDK and contains example code on how to create layouts and programmed events.
To understand the contents of this article, you should know the basics of Android programming.

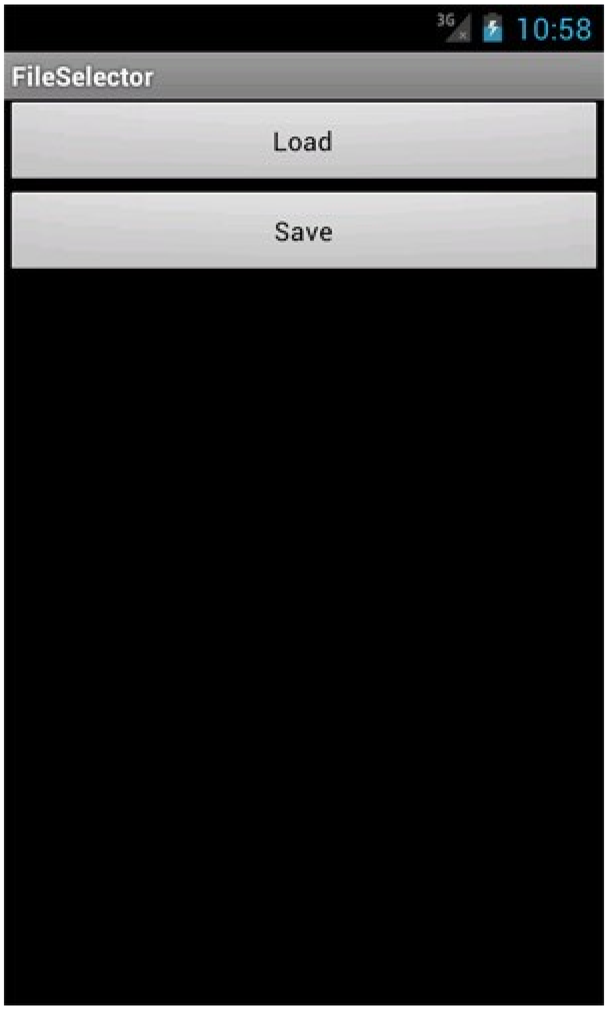
Creating the app

1. Create an activity

To begin, we need to create an activity that calls the control. The FileselectorActivity contains two buttons, one to load a file and the second to save the file. The buttons also need an OnClickListener. In the onClick() method, create and show the FileSelector control. The following code shows an example of implementing the settings listener:

```
mSaveButton.setOnClickListener(new OnClickListener() {

              @Override

              public void onClick(final View v) {

      FileSelectorActivity fs =

      new FileSelector(FileSelectorActivity.this,
      FileOperation.SAVE, mSaveFileListener, mFileFilter);

      fs.show();

      }
});
```

[Code 01] example – setting the OnClickListener



[Image 1] Sample FileselectorActivity

Create the control

- The control layout

The control layout is defined in the .xml file. The FileSelector uses components such as a ListView, Button, Spinner and an EditText. The ListView is needed to display a list of files, the Spinner is used to define the file's filter, Buttons define actions such as "Load", "Save", "New Folder" (this action creates a new folder in the current folder) and "Cancel".

Below is the code that defines the FileSelector layout.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/saveFileDialog"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/fileList"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.5"
        android:padding="10dp" />

    <ScrollView
                android:layout_width="fill_parent"
                      android:layout_height="wrap_content"
                      android:layout_weight="0.5">

                  <LinearLayout
                      android:id="@+id/fileLinearLayout2"
                      android:layout_width="fill_parent"
                      android:layout_height="wrap_content"
                      android:orientation="vertical"
                      android:padding="10dp" >

                      <TextView
                          android:id="@+id/fileTextView1"
                          android:layout_width="wrap_content"
                          android:layout_height="wrap_content"
                          android:padding="10dp"
                          android:text="@string/enterFileName"
                   android:textAppearance=
                        "?android:attr/textAppearanceMedium" />

                      <EditText
                          android:id="@+id/fileName"
                          android:layout_width="fill_parent"
                          android:layout_height="wrap_content"
                          android:contentDescription="@string/enterFileName"
                          android:inputType="text"
                          android:padding="10dp" />

                      <LinearLayout
                          android:id="@+id/fileLinearLayout3"
                          android:layout_width="fill_parent"
                          android:layout_height="wrap_content"
                          android:orientation="horizontal" >

                          <Button
                              android:id="@+id/fileSaveLoad"
                              android:layout_width="fill_parent"
                              android:layout_height="wrap_content"
```

```
                                android:layout_weight="1"
                                android:padding="10dp"
                                android:text="@string/saveButtonText" />

                            <Button
                                android:id="@+id/fileCancel"
                                android:layout_width="fill_parent"
                                android:layout_height="wrap_content"
                                android:layout_weight="1"
                                android:padding="10dp"
                                android:text="@string/cancelButtonText" />

                            <Button
                               android:id="@+id/newFolder"
                               android:layout_width="fill_parent"
                               android:layout_height="wrap_content"
                               android:layout_weight="1"
                      android:text="@string/newFolderButtonText"

                   />

                        </LinearLayout>

                        <Spinner
                                  android:id="@+id/fileFilter"
                                  android:layout_width="fill_parent"
               android:layout_height="wrap_content" />

                   </LinearLayout>

   </ScrollView>

</LinearLayout>
```

[Code 02] dialog.xml

- The control class

The class needs variables that will represent the activity controls, so setting up the FileSelector and the data that it uses is essential.

The FileSelector is represented by means of a Dialog class. The dialog contains a list of files, a box with the selected file name, and buttons: Save or Load and Cancel. The dialog and its elements are prepared in the FileSelector constructor.
In the first step we must prepare an object of the Dialog type:

```
mDialog = new Dialog(context);
```

```
 mDialog.setContentView(R.layout.dialog);
 mDialog.setTitle(mCurrentLocation.getAbsolutePath());
```

[Code 03] prepare the Dialog

The Dialog uses a layout defined in the xml file.
In the next steps, the dialog items are prepared.

The list of files.

The list of files is presented by a ListView. The following example shows how this can be achieved.

```
private void prepareFilesList() {

mFileListView = (ListView)mDialog.findViewById(R.id.fileList);

mFileListView.setOnItemClickListener(new OnItemClickListener() {

                              @Override
              public void onItemClick(final AdapterView parent, final
              View view, final int position, final long id){

                                             // This code will be call on click action

                                             }

              });

              String filtr = mFilterSpinner.getSelectedItem().toString();
makeList(mCurrentLocation, filtr);

}
```

[Code 04] prepareFilesList() code

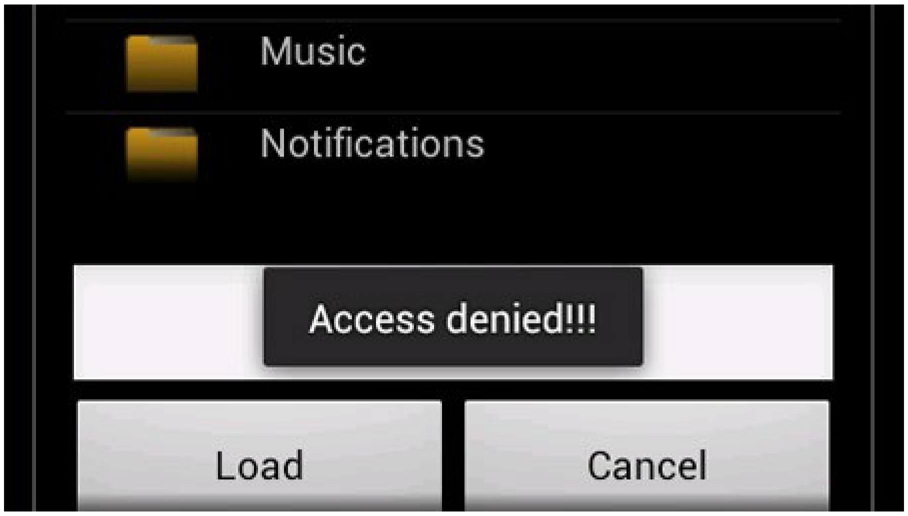
First, assign the view defined in the .xml to a variable. Then attach an OnItemClickListener and create a list by using the makeList(current_location, filter) method. When selecting a list item, the following code is executed.

```
((EditText) mDialog.findViewById(R.id.fileName)).setText("");
if (id == 0) {
              final String parentLocation = mCurrentLocation.getParent();
              if (parentLocation != null) { // text == "../"
                              String fileFilter = ((TextView)
                              FilterSpinner.getSelectedView()).getText().toString();
                              mCurrentLocation = new File(parentLocation);
                              makeList(mCurrentLocation, fileFilter);
              } else {
```

```
                              onItemSelect(parent, position);
                              }
} else {
              onItemSelect(parent, position);
}
```

[Code 05] method body of mFileListView.OnItemClickListener{...}.onItemClick() {...}

In the method “onItemSelect(parent)”, if the selected item is read-only, display a message and leave the function, as shown in the following image.



[Image 2] Message: “Access denied!!!”

If it is a folder, a new list is created, if it is a file its name is inserted into a text box, as shown in the following code.

```
private void onItemSelect(final AdapterView parent, final int position) {
              final String itemText = ((FileData)
parent.getItemAtPosition(position)).getFileName();
              final String itemPath = mCurrentLocation.getAbsolutePath() +
File.separator + itemText;
              final File itemLocation = new File(itemPath);
              if (!itemLocation.canRead()) {
                              Toast.makeText(mContext, "Access denied!!!",
                                                   Toast.LENGTH_SHORT).show();
              } else if (itemLocation.isDirectory()) {
                              mCurrentLocation = itemLocation;
                              String fileFilter = ((TextView)

mFilterSpinner.getSelectedView()).getText().toString();
                              makeList(mCurrentLocation, fileFilter);
              } else if (itemLocation.isFile()) {
```

```
                              final EditText fileName = (EditText)

mDialog.findViewById(R.id.fileName);
                              fileName.setText(itemText);
              }
}
```
[Code 06] onItemSelect(…) code

The makeList method creates an ArrayList. This method gets a list of files to display, creates filters and sorts the collection, then sets an adapter on the ListView, as shown in the following code.

```
private void makeList(final File location, final String filter) {
         final ArrayList<FileData> fileList = new           ArrayList<FileData>();
         final String parentLocation = location.getParent();
         if (parentLocation != null) {
                  fileList.add(new FileData("../", FileData.UP_FOLDER));
         }
         File listFiles[] = location.listFiles();
         if (listFiles != null) {
                  ArrayList<FileData> fileDataList = new
                  ArrayList<FileData>();
                  for (int index = 0; index < listFiles.length; index++)   {
                           File tempFile = listFiles[index];
                           if (FileUtils.accept(tempFile, filter)) {
                                    int type = tempFile.isDirectory() ?
                                    FileData.DIRECTORY : FileData.FILE;
                                    fileDataList.add(new
FileData(listFiles[index].getName(),
                                    type));
                           }
                  }
                           fileList.addAll(fileDataList);
                           Collections.sort(fileList);
                  }
                           if (mFileListView != null) {
                           FileListAdapter adapter = new FileListAdapter(mContext,fileList);
                           mFileListView.setAdapter(adapter);
                  }
}
```
[Code 07] makeList(…) code

This example uses an ArrayList of FileData. This object type has two members: one String – the file name, and an integer – mFileType (Must use one of these constants: UP_FOLDER, DIRECTORY, FILE). The File Data object must implement the Comparable interface.

In order for the ListView object to work properly, it needs an adapter. The example above uses a FileListAdapter. The FileListAdapter object inherits from the BaseAdapter class. In the constructor of the adapter two objects are transferred: a Context and a collection of FileData objects. The getCount() method returns the collection's size. The getItem(int position) method returns an array element of [position] index. The getItemId(int position) method returns the item's position number. The getView(int position, View v, ViewGroup parent) method returns a TextViewWithImage object. This object consists of an ImageView and a TextView in a LineralLayout with horizontal orientation. A context is passed to the constructor, and the class fields are prepared, as shown in the following code.

```
public TextViewWithImage(Context context) {
              super(context);
              setOrientation(HORIZONTAL);
              mImage = new ImageView(context);
              mText = new TextView(context);
              LayoutParams lp = new LayoutParams(0,
              LayoutParams.WRAP_CONTENT, 1);
              lp.weight = 1;
              addView(mImage, lp);
              lp = new LayoutParams(0, LayoutParams.WRAP_CONTENT, 3);
              addView(mText, lp);
}
```

[Code 08] TextViewWithImage(…) constructor code

The TextViewWithImage object has three methods. First, getText() returns an empty string if the TextView is null, or the value of the getText() method from the TextView, if the object is not null. Second, setImageResource(int resId) sets an image from the resources for the ImageView, but if resId is equal to -1, it sets the ImageView's visibility to GONE. Finally, setText(String aText) sets the text for the TextView.

Save\Load button.
The Save\Load button is prepared in setSaveLoadButton(FileOperation operation). This code shows the method body.

```
private void setSaveLoadButton(final FileOperation operation) {
              mSaveLoadButton = (Button)
              mDialog.findViewById(R.id.fileSaveLoad);
              switch (operation) {
                             case SAVE:
                             mSaveLoadButton.setText(R.string.saveButtonText);
                             break;
                             case LOAD:
                             mSaveLoadButton.setText(R.string.loadButtonText);
                             break;
              }
              mSaveLoadButton.setOnClickListener(new
              SaveLoadClickListener(operation, this, mContext));
}
```

[Code 09] setSaveLoadButton(...) constructor code

FileOperation is an enum with two options: SAVE and LOAD. The SaveLoadClickListener is a class which implements an OnClickListener. The class constructor receives the type of operation, a FileSelector object and a context. In the onClick(View v) method, if relevant conditions are fulfilled, called the handleFile(filePath) method from the OnHandleFileListener object, as shown in the following code.

```
public void onClick(final View view) {
              final String text = mFileSelector.getSelectedFileName();
              if (checkFileName(text)) {
              final String filePath =
              mFileSelector.getCurrentLocalizzation().getAbsolutePath() +
              File.separator + text;
              final File file = new File(filePath);
              int messageText = 0;
              switch (mOperation) {
                              case SAVE:
                              if ((file.exists()) && (!file.canWrite())) {
                                             messageText = R.string.cannotSaveFileMessage;
                              }
                              break;
                              case LOAD:
                              if (!file.exists()) {
                                             messageText = R.string.missingFile;
                                             } else if (!file.canRead()) {
                                             messageText = R.string.accessDenied;
                              }
                              break;
              }
              if (messageText != 0) {
                              final Toast t = Toast.makeText(mContext,
messageText,Toast.LENGTH_SHORT);
                              t.setGravity(Gravity.CENTER, 0, 0);
                              t.show();
              } else {
              mFileSelector.mOnHandleFileListener.handleFile(filePath);
              mFileSelector.dismiss();
                              }
              }
}
```

[Code 10] onClick(...) code

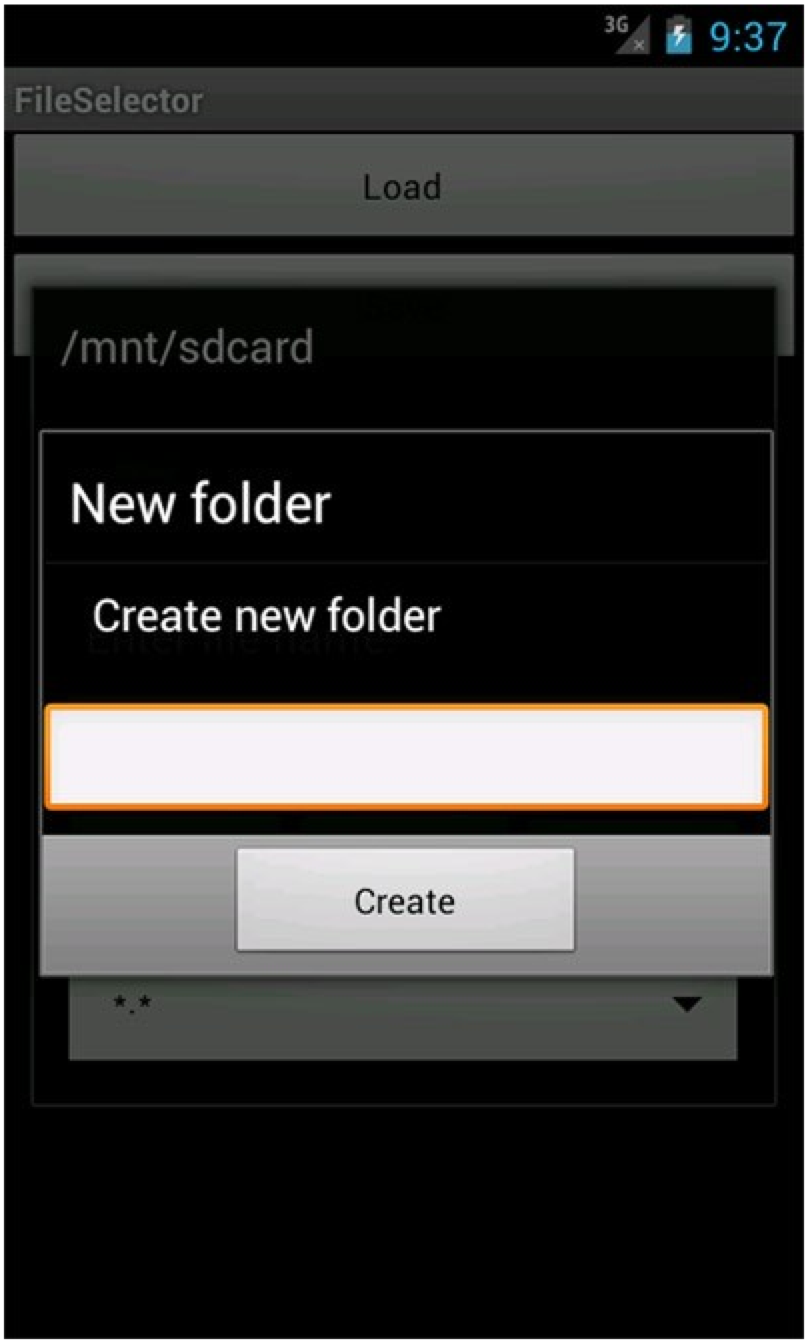
The checkFileName(String aText) method checks that the file name is in the EditText, if not, it shows the appropriate information.
OnHandleFileListener is an interface that enables communication between the FileSelector and the Activity. The path to the selected file is passed in the handleFile(String filePath) method. Create an OnHandleFileListener object in FileselectorActivity and pass it to the FileSelector constructor.

New Folder button.
The New Folder button creates a new folder in the current folder. This button is prepared in setNewFolderButton(FileOperation o) , as shown in the following code.

```
private void setNewFolderButton(final FileOperation operation) {
              mNewFolderButton = (Button)
              mDialog.findViewById(R.id.newFolder);
              OnClickListener newFolderListener = new OnClickListener() {
                             @Override
                             public void onClick(final View v) {
                                            openNewFolderDialog();
                             }
              };
              switch (operation) {
                             case SAVE:
                             mNewFolderButton.setVisibility(View.VISIBLE);
                             mNewFolderButton.setOnClickListener(newFolderListener);
                             break;
                             case LOAD:
                             mNewFolderButton.setVisibility(View.GONE);
                             break;
              }
}
```

[Code 11] setNewFolderButton(...) code

The button is visible only in SAVE mode. The openNewFolderDialog() method opens a dialog for creating a new folder.
The following image shows a sample dialogue.



[Image 3] Sample dialogue to create new folder

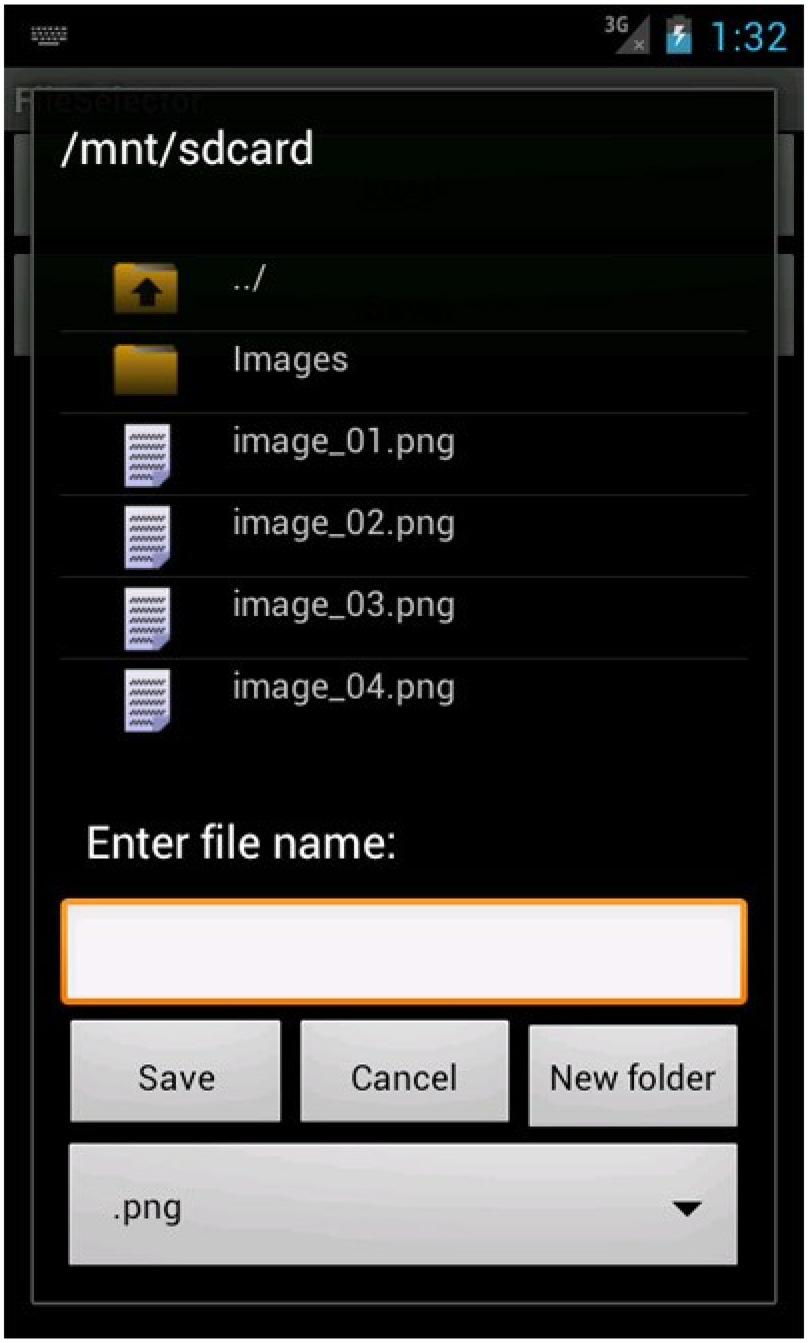
Filter spinner

The Filter spinner is prepared in prepareFilterSpinner(String[] filters).This method should be called first. This method sets the adapter, and the OnItemSelectedListener. If the filters array is null or empty, the filter FILTER_ALLOW_ALL ("*.*") is added to the spinner and the "enabled" parameter is set to false. This code shows the body of the method.

```
private void prepareFilterSpinner(String[] fitlesFilter) {
  mFilterSpinner = (Spinner) mDialog.findViewById(R.id.fileFilter);
  if (fitlesFilter == null || fitlesFilter.length == 0) {
    fitlesFilter = new String[] {
      FileUtils.FILTER_ALLOW_ALL

    };
    mFilterSpinner.setEnabled(false);
  }
  ArrayAdapter adapter = new ArrayAdapter(mContext, R.layout.spinner_item, fitlesFilter);
  mFilterSpinner.setAdapter(adapter);
  OnItemSelectedListener onItemSelectedListener = new OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView aAdapter, View aView, int arg2, long arg3) {
      TextView textViewItem = (TextView) aView;
      String filtr = textViewItem.getText().toString();
      makeList(mCurrentLocation, filtr);
    }
    @Override
    public void onNothingSelected(AdapterView arg0) {
    // do nothing
    }
  };
  mFilterSpinner.setOnItemSelectedListener(onItemSelectedListener);
}
```

[Code 12] prepareFilterSpinner(...) code

The final score shows the following image:



[Image 4] FileSelector

ref:http://developer.samsung.com/android/technical-docs/Implementing-a-file-selector-dialog