File Transfer Module Usage Guide for Android

HT80-BA067-1 Rev. A

March 2013

Submit technical questions at:

http://www.alljoyn.org/forums

Qualcomm Innovation Center, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Contents

1 Introduction

1.1 Scope

This document is a guide to using the AllJoyn File Transfer module on Android devices. It covers the following topics:

- Setting up the Eclipse project
- Adding the module to your application
- Using the module, including code snippets
- Unit tests

1.2 Audience

This guide is for AllJoyn Android developers who wish to add to their application the ability to transfer files between two AllJoyn Peers.

1.3 Prerequisites

- Before proceeding with development as described in this document, set up the development environment as described in the *AllJoyn Android Environment Setup Guide*.
- It is recommended that you read the *Guide to AllJoyn Development Using the Java SDK*.

1.4 Related documents

Related documents is a list of supplemental documents and downloads that relate to the information in this guide.

Table 1: Related documents

Title	Location
Guide to AllJoyn Development Using the Java SDK	*http://www.alljoyn.org*
AllJoyn Android Environment Setup Guide	*http://www.alljoyn.org*

1.5 Revision history

Revision history provides the revision history for this document.

Table 2: Revision history

Revision	Date	Description
A	March 2013	Initial release

2 Overview

The File Transfer Module (FTM) is a library that provides application developers with a tool they can use to easily transfer files between peers over AllJoyn. The application developer can use the API for simple file transfers, where files are announced to session peers and can then be requested for transfer. The developer can also use the more advanced file transfer mechanisms, which allow a peer to explicitly offer files to other peers (a virtual push), or allow a peer to request files that have not been announced.

The FTM also supports dynamic sessions, which allow the user to specify multiple instances of the FTC and provide a different AllJoyn session to each instance. Furthermore, the user does not even have to give the FTM an AllJoyn session to work with. Most of the file transfer operations can still be used without an AllJoyn session by the user. For example, the user can still announce files but the announcements will not be sent over AllJoyn until a session is specified. In such a case each file transfer operation will stop short of sending any signals or performing method calls over AllJoyn.

This document describes the procedure to download the FTM source code, build the project, and use the software in your application. The Javadocs are intended to provide a detailed description of the File Transfer API methods. For more information regarding the API, see *Regenerating the Javadocs*. For more information on the AllJoyn peer-to-peer communications framework, go to *http://www.alljoyn.org*.

3 Setup and Installation

3.1 Setup

Download and Install:

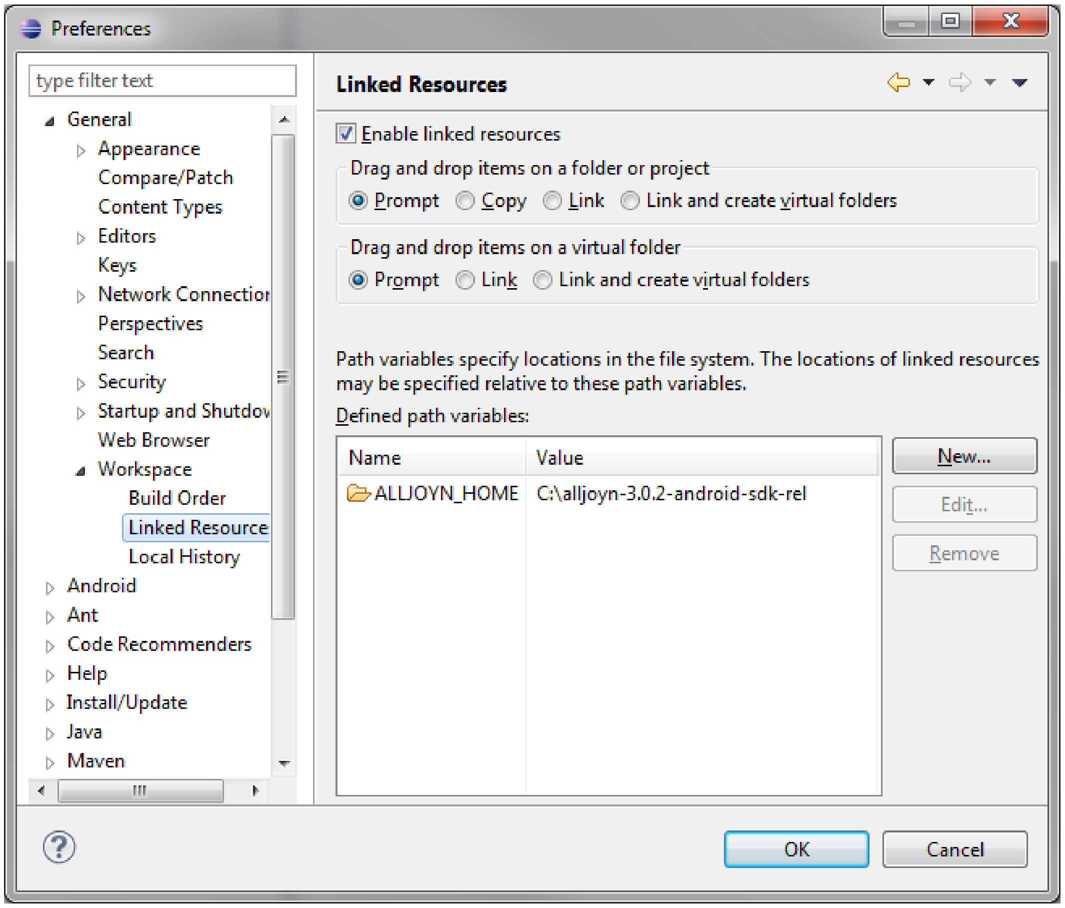
- Eclipse
- ADT Plugin for Eclipse
- Android SDK
    - API Level 8+ (Android 2.2+)
- AllJoyn Android SDK
    - Release 2.3.3 or higher
- Java JDK
    - Version 1.6
    - This is used to generate the Javadocs

Instructions can be found at *https://www.alljoyn.org/docs-and-downloads*

3.2 Building the FileTransferComponent project

3.2.1 Importing the FileTransferComponent into Eclipse

1. On the top menu, select **File > Import**
2. Select **General > Existing Projects into Workspace** and click **Next.**
3. Click **Browse** next to the **Select root directory** field and navigate to the **FileTransferComponent** eclipse project within the Git repository that you cloned.
4. Verify that the **Copy projects into workspace** option is unchecked, then click **OK** and **Finish**.
5. On the top menu, select **Window > Preferences**.
6. On the left hand side, select **General > Workspace > Linked Resources** as shown below:



Linked Resources

[✓] Enable linked resources

Drag and drop items on a folder or project

[✓ Prompt] Copy Link Link and create virtual folders

Drag and drop items on a virtual folder

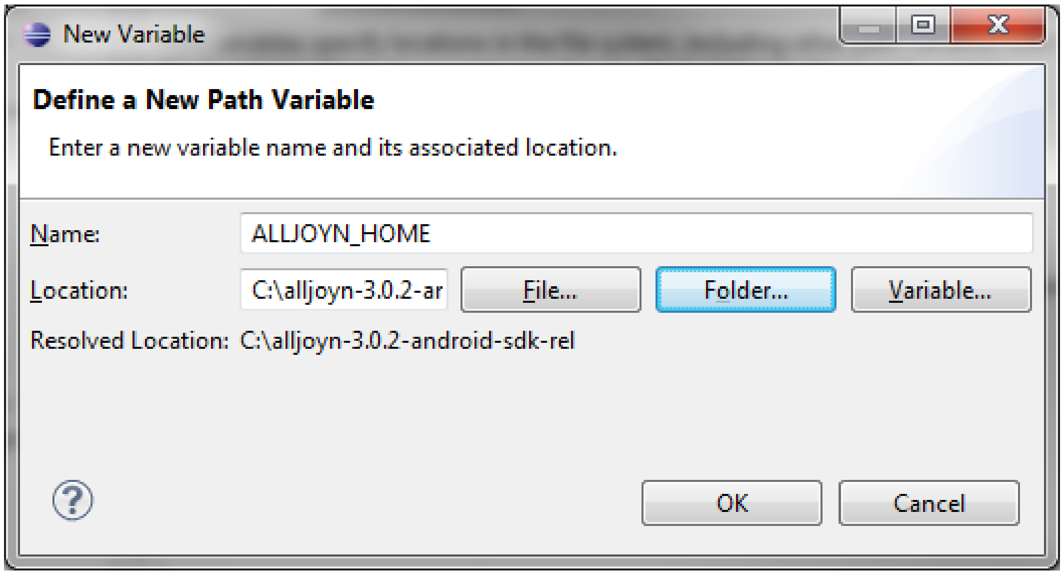
[✓ Prompt] Link Link and create virtual folders

Path variables specify locations in the file system. The locations of linked resources may be specified relative to these path variables.

Defined path variables:

Name	Value
ALLJOYN_HOME	C:\alljoyn-3.0.2-android-sdk-rel

7. Notice that there is a path variable called ALLJOYN_HOME that points to the root folder of the AllJoyn Android SDK. You need to create this same variable in your environment, and set it to the path where you saved the AllJoyn Android SDK.
   a. Make sure the **Enable linked resources** option is checked, and then click **New…** to add a new Path Variable.
   b. A window should appear to allow you to create a new Path Variable. The name should be “ALLJOYN_HOME” and the path should point to the location of the root folder for the AllJoyn Android SDK. An example is shown below:


New Variable

Define a New Path Variable

Enter a new variable name and its associated location.

Name: ALLJOYN_HOME

Location: C:\alljoyn-3.0.2-ar

Resolved Location: C:\alljoyn-3.0.2-android-sdk-rel

c. Click **OK** to create the new Path Variable. You should see it added to the list of defined path variables.

d. Click **OK** to close the Preferences window.

3.2.2 Regenerating the FileTransferComponent JAR

The project is marked as an Android library and the jar is located in the `bin` directory of the project. If you have the automatic build feature enabled in Eclipse under **Project > Build Automatically**, the project is rebuilt and the jar is updated every time changes are made to the source code and you hit **Save.**

3.2.3 Regenerating the Javadocs

1. Select the FileTransferComponent project in the Package Explorer.
2. From the top menu, select **Project > Generate Javadoc…**
3. Click **Configure…** next to the Javadoc command field and navigate to the `javadoc.exe` within your Java JDK installation.

   An example path is `C:\Program Files (x86)\Java\jdk1.6.0_35\bin\javadoc.exe`
4. Expand the drop-down menu for the FileTransferComponent in the **Select types for which Javadoc will be generated** field and uncheck the `gen` directory, leaving only the `src` directory checked.
5. Click **Finish** and the Javadocs will be generated in the `doc` directory of the FileTransferComponent project.

Note Sometimes if APIs are removed from the code, they will still appear in the Javadocs. In this case, delete all of the old Javadocs in the `docs` directory and regenerate everything.

3.3 Adding the File Transfer Module to your application

Once you have imported the FileTransferComponent project into Eclipse and successfully built the JAR file, there are two distinct methods to add this functionality to your application:

- Add just the FileTransferComponent JAR file into the build path libraries that are used by your project, or
- Reference the entire FileTransferComponent project as a library project.

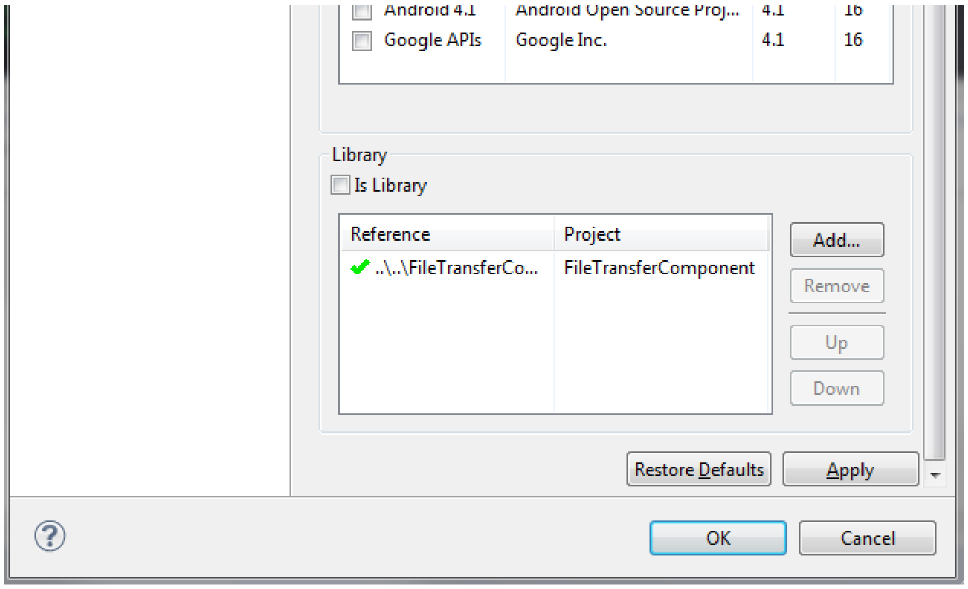
You only need to choose one of these methods, described in the following sections.

3.3.1 Adding the FileTransferComponent JAR to your project

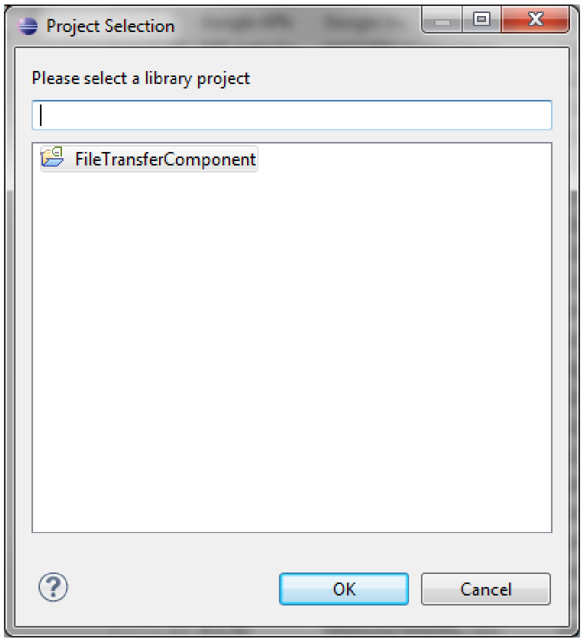
1. Copy the `filetransfercomponent.jar` from `FileTransferComponent/bin` into the `libs` directory of your Android project.
2. Right-click your project in the package explorer of Eclipse and select **Build Path > Configure Build Path…**
3. Select the **Libraries** tab and click **Add JARs…**.
4. Navigate to the `filetransfercomponent.jar` in the `libs` directory of your project. Click **OK** and **OK** again.

3.3.2 Referencing the FileTransferComponent Project in your application project

1. Right-click your project and select **Properties**.
2. Select **Android** from the menu on the left side. A window will appear that allows you to choose the Project Build Target and, at the bottom, reference an existing project as shown below:



3. In the screenshot, this project already references the FileTransferComponent project as a library project.
4. Click the **Add…** button to add a new library project reference. The following menu appears:



5. The FileTransferComponent appears by default because it is configured to be used as a library.
6. Select the FileTransferComponent and click **OK**. You should see the project appear as a reference with a green check mark, as shown in Step 2.
7. Click **OK** to close the project properties menu.

3.4 File Transfer Module unit tests

The file transfer unit tests project contains the test code written to ensure that the FileTransferComponent functions correctly. If you intend to modify the FileTransferComponent source code but want to preserve its original behavior, running these tests can help verify that the behavior has not changed. Please note, however, that passing the unit tests does not necessarily mean that your changes will function correctly.

3.4.1 Importing the FileTransferComponent unit tests into Eclipse

1. On the top menu, select **File > Import**.
2. Select **General > Existing Projects into Workspace** and click **Next**.
3. Click **Browse** next to the **Select root directory** field and navigate to the FileTransferUnitTests eclipse project, under the test directory, within the Git repository that you cloned. Click **OK** and **Finish.**

3.4.2 Running the FileTransferComponent unit tests

1. Select the FileTransferUnitTests project in the Eclipse package explorer.
2. On the top menu, select **Run > Run As > Android JUnit Test**.

This process will use the Android emulator if you do not have an Android device connected to your computer.

3.5 File Transfer Module sample Android application

The file transfer module sample Android application is intended to provide the user with a glimpse into how to use the FileTransferComponent in a custom Android application. The sample application is very useful in showing how to use the core file transfer functionality inside an Android AllJoyn application.

Note The file transfer module sample application explicitly references the FileTransferComponent project as a library and does not import the jar file. See *Referencing the FileTransferComponent Project in your application project*.

Feel free to modify the sample application code, as it will help in your understanding of how to use the FileTransferComponent.

3.5.1 Importing the FileTransferComponent sample application into Eclipse

1. On the top menu, select **File > Import**.
2. Select **General > Existing Projects into Workspace** and click **Next**.
3. Click **Browse** next to the **Select root directory** field and navigate to the FileTransferSampleApp eclipse project, under the samples directory, within the Git repository that you cloned. Click **OK** and **Finish.**

3.5.2 Running the FileTransferComponent sample

After importing the file transfer sample application into eclipse, you can run the application on your Android device.

Note If you are unfamiliar with how to install Android applications on devices, refer to the documentation on the Android Developers site *http://developer.android.com*

4 Usage

4.1 File transfer methods

4.1.1 Basic File Transfer procedures

A basic transfer requires a session peer (sender) to announce that files are available for transfer, and then some other session peer (receiver) to request one or more of them. This is the traditional pull method of the file transfer module. The following steps outline the minimum steps needed to transfer files using AllJoyn:

1. The sender announces files using the announce(…) method, passing a list of one or more absolute file paths as a parameter. This function will send an announcement to the session peers to indicate that these files are available for transfer.
2. The receiver can obtain a list of all files announced by sender(s) and will use this list to send file requests to a specified sender, using the file ID and file name.

Sender Side

```
ftComponent.announce(filePaths);
```

Receiver Side

```
ArrayList<FileDescriptor> availableFiles = ftComponent.getAvailableRemoteFiles();
FileDescriptor selected = availableFiles.get(0);
ftComponent.requestFile(selected.owner, selected.fileId, selected.filename);
```

4.1.2 Offer file

Alternatively, the offer file transfer method can be used to “push” a file to a peer. The peer, however, must accept the offer before the transfer can take place. This is accomplished through the OfferReceivedListener. By default, all offers will be refused. If you wish to use the offer file method, you must follow the steps outlined below to ensure correct functionality:

1. Register the `OfferReceivedListener` on the receiver. Override the listener’s `acceptOfferedFile(…)` method to return true to allow the transfer.
2. Sender calls `offerFileToPeer(…)` providing the receiver’s unique bus id, the file path of the offered file, and a timeout. `OfferFileToPeer` will block the calling thread until the offer is either accepted, rejected, or the timeout interval is exceeded.

   OfferFileToPeer is implemented using AllJoyn over-the-air methods, meaning the offer must be accepted or denied quickly or it will time out. UI interaction inside the listener is not recommended.

Receiver Side

```
ftComponent.setOfferReceivedListener(new OfferReceivedListener()
{
public boolean acceptOfferedFile(FileDescriptor file, String peer)
{
return true;
}
});
```

Sender Side

```
ftComponent.offerFileToPeer(peer, filePath, 1000);
```

4.1.3 Request offer

If a file resides on a remote peer at a known file path, the `requestOffer` method can be used by the receiver to request the file using the absolute path. Files requested this way do not need to be previously announced or shared. The `UnannouncedFileRequestListener` is used by the sender to guard against unwanted transfers. If the listener is not registered, the request will be denied by default. If the listener is registered, it will be queried to see if the request should be granted. If you wish to use the request offer transfer method, the following steps must be followed to ensure correct functionality.

1. Register the `UnannouncedFileRequestListener` on the sender using the `setUnannouncedFileRequestListener(…)` method.
2. Override the listener's `allowUnannouncedFileRequests(…)` method to return true to allow the transfer.
3. Register the `FileAnnouncementReceivedListener` on the receiver using the `setFileAnnouncementReceivedListener(…)` method. If the transfer is allowed, this listener will be called with the desired file's descriptor (containing the file ID) as a parameter.
4. On the receiver, call `requestOffer(…)` providing the remote file owner's unique bus ID and the absolute file path. If the request is granted, the receiver will receive a directed announcement, through the `FileAnnouncementReceivedListener`, containing the file descriptor for the requested file.

   Note `requestFile(…)` cannot be called inside the `FileAnnouncementListener` without first spinning up a new thread. This is because the listener is triggered by the AllJoyn thread, and AllJoyn does not allow remote calls to be made on its own thread. See the example below.

5. The receiver calls `requestFile(…)`, initiating the transfer.

Sender Side

```
ftComponent.setUnannouncedFileListener(new UnannouncedFileRequestListener()
{
 public boolean allowUnannouncedFileRequests(String filePath)
 {
  /*
  Note: an actual implementation would probably want to
  return false if filePath did not exist.
            */
  return true;
 }
});
```

Receiver Side

```
ftComponent.setFileAnnouncementReceivedListener(new
     FileAnnouncementReceivedListener()
{
 public void receivedAnnouncement(FileDescriptor[] fileList, boolean
     isFileIdResponse)
 {
  new Thread(new Runnable()
  {
   @Override
   public void run()
   {
    FileDescriptor file = fileList[0];

    ftComponent.requestFile(file.owner, file.fileId,       file.filename);
   }
  }).start();
 }
});

ftComponent.requestOffer(peer, path);
```

4.2 File Transfer utility functions

This section outlines some of the File Transfer API functions that will enhance your understanding of how to use all of the features of the API. This section does not cover all of the methods available in the API and does not comprehensively describe all functionality. For more information about each API method, download the source code and regenerate the Javadocs. This process is outlined in *Regenerating the Javadocs*.

4.2.1 Initialize

This function provides a mechanism to allow the user to specify a new AllJoyn session with the current instance of the File Transfer Module. The provided AllJoyn bus attachment and session ID are used to create a new file transfer bus object and the objects are passed down into the necessary file transfer modules so the new AllJoyn session can be used to transfer files.

4.2.2 Uninitialize

This function allows the user to disassociate the originally specified AllJoyn session with the current instance of the File Transfer Module. The user does not have to specify a new AllJoyn session to be used, since most file transfer operations can still be used. This gives the user tremendous flexibility to give an AllJoyn session to the File Transfer Module, when needed.

4.2.3 Stop Announce

This function provides a mechanism for the user to stop announcing files they do not wish to make available to session peers. This function takes an array of absolute file paths and will remove all files from the announced file list that match one of the provided absolute paths. Once files have been removed from the announced file list, a new file announcement is sent to all session peers showing only the files that are still available for transfer.

4.2.4 Request File Announcement

This function allows any session peer to request the announced files from session peers. This function is intended to be used when a user joins a session late because files have, for the most part, already been announced and it may take time for the new user to receive file announcements from session peers. Therefore, the new user can send a directed or global AllJoyn signal to one or all session peers indicating that they want a list of the files that have been announced.

Note The developer will need to register the File Announcement Received Listener for correct functionality. If this listener is not registered, the developer will not know when session peers have responded to their announcement request. For more information on how to register the announcement listener, see the section *Request offer*.

4.2.5 Get File ID

This function will search the list of available remote files for one that matches the provided path and peer parameters. If a match is found, the file ID is returned. If a match is not found, this function will return null.

4.2.6 Get Available Remote Files

A key piece to the File Transfer API is the ability to maintain a local list of all the files that have been made available for transfer by session peers. This list of all available files can be returned by calling this function. This allows the developer to maintain an accurate list of the files available for transfer. This list will include files that have been accumulated through formal announcements, as well as files accumulated through informal channels, such as file offers and offer requests.

4.2.7 Get Announced Local Files

This function allows the developer to see an updated list of all files that he has currently made available to session peers for transfer. This function makes it easier to see a complete list of all announced files and is intended to be utilized when the user wishes to stop announcing a subset of their announced files.

4.2.8 Get Offered Local Files

The offered file list is added to only when the user offers a file directly to a session peer or allows a session peer to request a file that has not been announced. Offered files are kept separate from announced files so only the peer that received the offer or sent the offer request can receive the file. This function allows the user to see which files have been shared to one or more session peers. This function does not allow you to modify the offered file list.

4.2.9 Set Show Relative Path

This function allows the user to specify their wish for their remote peers to see the relative path of their announced files. This value is true by default.

4.2.10 Get Show Relative Path

This function will return to the user the current value of the `showRelativePath` setting.

4.2.11 Clean Cache

To prevent the cache file from growing without limits, the user can call this function to remove any hash values that point to files that either do not exist or have been modified since their hash value was calculated.

4.2.12 Set Show Shared Path

This function allows the user to specify whether or not the wish for their remote peers to see the shared path of their announced files. This value is false by default.

4.2.13 Get Show Shared Path

This function will return to the user the current value of the `showSharedPath` setting.

4.2.14 Set Default Save Directory

This method allows the user to specify the directory where all transferred files will be stored.

4.2.15 Set Cache File

This function allows the user to pass a path to a file or a file object that will be used to store the hash values of all announced/offered files. Since hashing each file can be a computationally-expensive task, caching will help cut down that time by reusing the value for files that have already been announced/offered. Obviously, caching will be beneficial only if you are sharing the same files multiple times.

4.2.16 Cancel Sending File

This function allows the file sender to cancel the file transfer that matches the specified file ID. After the cancel is initiated, a cancel notification is sent to the file receiver stating that the sender has terminated the file transfer. The receiver will maintain all temporary files so the transfer can be resumed at a later time.

4.2.17 Cancel Receive File

This function allows the receiver to cancel the file transfer that matches the specified file ID. After the transfer is cancelled, all temporary files are deleted and a notification is sent to the sender to stop sending the remaining file data. Since all temporary files are deleted, if the user wishes to request the file again, the transfer will have to start from the beginning.

4.2.18 Pause File

Pause file can be called only by the file transfer receiver and allows the receiver to pause a file transfer and to resume it at a later time. All temporary files are maintained in memory.

4.2.19 Get Sending Progress List

This function allows the user to monitor the progress of all files they are sending to session peers. When this function is called, the user will receive a list of files that are being transferred to session peers and also a list of the number of bytes that have been sent for each file.

4.2.20 Get Receive Progress List

This function allows the user to monitor the progress of all files they are receiving from session peers. When this function is called, the user will receive a list of files they are currently receiving from session peers and also a list of the number of bytes that have been received for each file.

4.2.21 Destroy

Once you have finished using an instance of the File Transfer Module, you should call its `destroy()` method to clean up any resources it has allocated. This is especially important if you are creating many File Transfer Module instances, regardless of whether they are created all at once or over a period of time. After calling `destroy()`, you should not make any further calls on that instance, because that would be an error. However, you can continue to use other `FileTransferComponent` instances you have created, as well as instantiate new ones.

Sample Code for use of Destroy method

```
// Create a FileTransferComponent instance

FileTransferComponent ftc1 = new FileTransferComponent(busAttachment,
sessionID);

// Use the FileTransferComponent instance

ftc1.announce(filesToAnnounce); // etc.

[…]

// Sometime later, when done with ftc1, make sure to destroy it to free
resources

ftc1.destroy();

ftc1 = null;
```

4.3 Listeners

This section describes and demonstrates how to use some of the listeners that are provided in the File Transfer API. The listeners mentioned below are not mandatory but they will allow better monitoring of the occurring file transfer events.

4.3.1 File Completed Listener

The file completed listener can be used so the file receiver can be sent a notification when the file transfer has completed. This listener is just an interface that has to be implemented by the class that wishes to receive the callback. You can register the listener in other ways but the following code demonstrates how to register the listener within a call to the API `setFileCompletedListener()` method:

```
FileTransferComponent ftComponent = new FileTransferComponent(...);

...

ftComponent.setFileCompletedListener(new FileCompletedListener()
{
    public void fileCompleted(String filename, int statusCode)
    {
        /*
        The user can add any code they need to respond accordingly.
        */
    }
});

...
```

Note This listener is not required and does not interfere with normal file transfer functionality.

4.3.2 Request Data Received Listener

The request data received listener can be utilized so that the file sender can be notified when a file request has been received. The callback will show only the name of the file being requested and not who made the file request. This listener is just an interface that has to be implemented by the class that wishes to receive the callback. You can register the listener in other ways but the following code demonstrates how to register the listener with a call to the API `setRequestDataListener()` method:

```
FileTransferComponent ftComponent = new FileTransferComponent(..., ...);

...

ftComponent.setRequestDataReceivedListener(new RequestDataReceivedListener()
{
    public void fileRequestReceived(String fileName)
    {
        /*
        This function will be called when a file request has been received.
        The user can add any code they need to respond accordingly.
        */
    }
});

...
```

Note This listener is not required and does not interfere with normal file transfer functionality.

4.3.3 File Announcement Sent

The file announcement sent listener can be used so that the user can be notified when an announcement has been sent to remote session peers. Announcing files is an asynchronous operation because the SHA-1 hash of the file contents must be calculated for every announced file. This process becomes computationally expensive with larger files (e.g., video). Therefore, the file announcement sent listener can be registered so the user can receive feedback about the success or failure of their file announcement. This listener will return an array of paths for those files that failed to successfully announce. You can register the listener in other ways but the following code demonstrates how to register the listener within a call to the API. `setFileAnnouncementSentListener()` method:

```
FileTransferComponent ftc = new FileTransferComponent(...);

...

ftc.setFileAnnouncementSentListener(new FileAnnouncementSentListener()
{
    public void announcementSent(ArrayList<String> failedPaths)
    {
        /*

        The user can add any code they need to respond accordingly.
        */
    }
});

...
```

Note This listener is not required and does not interfere with normal file transfer functionality.