JSR 75 File connection API

The FileConnection API gives you the opportunity to access the device file system including Memory Sticks. It gives the functionality to create, remove directories and files, list directory content, set permissions, get file information and perform I/O operations on files. Below are the important classes and interfaces:

- ConnectionClosedException is thrown when a method is invoked on a FileConnection while the connection is closed.
- FileSystemRegistry class is a central registry and has the ability to list the mounted roots with the method listRoots(). It also provides methods for registering listeners that are notified if file systems are added and removed during run time.
- The FileSystemListener interface is used for receiving status notification when adding or removing a file system root.
- The FileConnection interface is used to access directories and files on the device.
- It extends the Connection interface and holds a number of useful methods to create, remove directories and files, list directory content, set permissions, get file information and perform I/O operations on files.

System properties: To check if the the FileConnection API is implemented use the:

```
System.getProperty("microedition.io.file.FileConnection.version");
```

This should return "1.0" if it is implemented.

Using the FileConnection API

Since the FileConnection API uses the Generic Connection Framework for file system connectivity, the procedure to create a FileConnection is performed in the same way as create any GCF connection, with the only difference being the URL.

To create a FileConnection, use the Connector factory's open method which returns a Connection:

```
Connector.open(string URL);
```

Valid URLs:

For internal memory "file://localhost/c:/" or "file:///c:/".
For memory card "file://localhost/c:/" or "file:///e:/

So when accessing the other directory in the internal memory, create a FileConnection object using the URL below:

```
>FileConnection fc = (FileConnection)Connector.open("file:///c:/");
```

Since the mode parameter in Connector.open() method is not supported, both the read and write permissions are set when creating a FileConnection object.

When creating a FileConnection, the actual file or directory that the URL is pointing to doesn't have to exist, so opening a file/directory and creating a new file/directory is very similar. This is necessary when creating files and directories.

Remember to close the FileConnection object when you are done using it:

```
fc.close();
```

A FileConnection that is pointing to a non-existent file will not be able to perform any file or directory specific operations. For example, if you are trying to open an InputStream or an OutPutStream, a java.io.IOException will be thrown.

The only difference is when the FileConnection is open the method create() or mkdir() is invoked to create the file or directory on the file system.

You can always check if the file/directory exists:

```
fc.exists();
```


(C) Copyright Nokia 2014. All rights reserved.

If it doesn't exist, create a new file:

```
fc.create();
```

or create a directory:

```
fc.mkdir();
```

To delete a file or a directory:

```
fc.delete();
```

To list the content of a directory that your FileConnection is pointing to:

```
Enumeration e = fc.list();
while (e.hasMoreElements()) {
System.out.println(((String)e.nextElement()));
}
```

This method returns an enumeration of all directories and files that are present in the directory. Directories are denoted with a trailing slash "/" in their returned name. There is a second version of the list method that enables you to sort the content with a filter:

```
list(String filter, boolean includeHidden)
```

This methods return the content that is passing the filter, the filter could be "*.mp3" for only listing files that ends with .mp3. Use the attribute includeHidden to indicate whether files marked as hidden should be included or not in the list of files and directories returned. These methods return folders and files in a random order so you need to sort them manually.

The FileSystemRegistry has the static method listRoots() to list the available and supported mounted roots on the device. The list is returned as an Enumeration :

```
Enumeration e = FileSystemRegistry.listRoots();
while (e.hasMoreElements()) {
String rootName = (String)e.nextElement();
System.out.println("mounted root:"+rootName);
}
```

File I/O

If you are used to the GCF connection framework, you will be very familiar with reading from files and writing to files. To write to a file you need to get an OutputStream from the FileConnection object that points to the existing file:

```
OutputStream os = fc.openOutputStream()
os.write(new String("hello").getBytes());
os.close();
```

Or you can use the DataOutputStream to write primitive Java datatypes to a file:

```
int i = 1234;
DataOutputStream ds = fc. openDataOutputStream();
ds.writeInt(i);
```


(C) Copyright Nokia 2014. All rights reserved.

```
ds.close();
```

To read from a file you get the InputStream or a DataInputStream from a FileConnection object that is pointing to the existing file:

```
byte[] b = new byte[1024];
InputStream is = fc.openInputStream();
is.read(b);
is.close();
```

File and directory information

There are a number of methods in the FileConnection class that are used for retrieving information about the specific directory or file. Here are some of them:

Is the file/directory readable?

```
boolean canRead()
```

Is the file/directory writable?

```
boolean canWrite()
```

Returns the size of all the files in bytes in the directory. If the includeSubDirs is true the size of all the sub-directories are included.

```
long directorySize(boolean includeSubDirs) -
```

Returns the size of the file in bytes

```
long fileSize() -
```

Returns the date when the file/directory was last modified l ong lastModified() -

(C) Copyright Nokia 2014. All rights reserved.