Documentation

Thursday, August 19, 2010
15:31

Author: Wesley Ellis
Contact: wesleyjellis@gmail.com

com.anrl.micandroid

- LocalService.java
  - Purpose: Implements Android's take on background services and interfaces with ListenService.java
  - Public methods:
    - onBind: Can be used to provide a binder, but this service is to simple, returns null
    - onStartCommand: not accessed directly. Return START_STICKY because we want the service to keep running
    - onDestroy: not accessed directly either
  - Private methods:
    - Notifier: creates the notification you see in the notification area
    - cancelNotify: removes the notification
    - Start: if we have already started a listenservice thread, do nothing, otherwise start ListenService
    - Kill: if we have a thread, then stop it
  - Usage:
    - Starting:

      ```
      Intent i = new Intent(this, LocalService.class);
      startService(i);
      ```

    - Stopping

      ```
      Intent i = new Intent(this, LocalService.class);
      stopService(i);
      ```

- Messages.java
  - Purpose: the interclass communication is done with handlers and the message object they provide. These messages use integers as IDs and this class prevens the reuse of an ID
- MICAdevice.java
  - Purpose: Custom object that represents a single instance of a mica client. It can be virtual or not. Contains several fields
    - Name: the name of the client
    - Host name: the name of the device hosting it
    - Host: The android/java(?) object representing a bluetooth device.
      - Note: this might be changed to a string because android will construct this object from a string if asked
    - Hashed Value: The result of a hashreq is stored here
  - Methods:
    - Mostly just getters, fields are filled when the constructor is called
    - toString returns the text mica address name@host@address
    - legacyGetAddress returns just name@address. This is used because the original idman/MICA protocol only supports name@address
  - Usage: As an object representing a device
- ReturnObject.java
  - Purpose: Android has an odd way of helping you store states. You can store 1 object for later retrieval. I need to keep two objects, so I wrapped them in this object.
  - Usage: in WizardUI.java

com.anrl.micandroid.bluetooth

- BluetoothMessage.java
  - Purpose: Base class for anything that uses bluetooth. Contains methods for connecting, closing, reading and writing. All methods are static (and probably not threadsafe…)
  - Protected methods:
    - connecTo: Pass a bluetooth device and it will see if that device is listening for a MICA connection, and if so, open a socket
    - Close: it closes the socket…

        - Write: Pass a string to be written to the socket. Will turn the string into a byte array
        - Read: Returns a string reply from the other device. Does some basic parsing. (Checking for headers/footers)
    - Usage: never directly. It's methods are accessed via the 4 classes that extend it

- HashReq.java
    - Purpose: provide a static method for getting hashes from a set of devices
    - Methods:
        - talkTo : creates a thread that passes back devices to a handler. The type of message distinguishes is if was a successful hash or not. It also passes back the MICA object with the hash set if it was succesful
    - Usage: Call talkto
- ListenService.java
    - Purpose: Contains the code to act as a MICA client. LocalService is essentially a wrapper for this class and its thread.
    - Public methods:
        - startBroadcast: pass the bluetooth adapter and the application context and the thread will loop until you call stopBroadcast
        - stopBroadcast: closes the serversocket in the thread and tells the thread to stop
    - Private methods:
        - Btthread.run: this thread sets up a server socket and then loops. It waits until a client connects to it, then reads in the message, and sends its reply.
        - Write: handles the writing to the socket
        - Parse: takes the input and figures out what to reply with
    - Usage: never directly, check out LocalService.java
- ListReq.java
    - Purpose: static method for discovering idman/MICA devices
    - Methods:
        - talkTo: creates a thread that passes back fully formed MICAdevices if they respond properly
    - Usage: call talkTo with a set of bluetooth devices.
- RegReq.java
    - Purpose: static method for sending out a register request. It's easier to
    - Methods:
        - talkTo: creates a thread that contacts a MICA device and sends a regreq to it
    - Usage: call talkTo with a set of MICA devices
- UnRegReq
    - Purpose: static method for unregistering a set of MICA devices
    - Methods:
        - talkTo: creates a thread that passes back a list of succesfully unregistered devices
    - Usage: call talkTo with a set of MICA devices

com.anrl.micandroid.checkboxlist

- CheckBoxAdapter.java
    - Purpose: Provide a UI element that will contain a list of textviews and checkboxes
    - Methods:
        - addItem: can take either a checkboxtitem or a MICAdevice and checked state. Will create an entry in the listview for it
        - setListItems: set the contents of the adapter to a pre made set. Probably because we're restoring after an onPause
        - getListItems: get the contents of the adapter
        - getCheckedItems: return a Set of MICA device objects that have a checkbox next to them.
        - getCount: return count
        - getItem: return the item at the specified ID
        - getView: this is called when each entry is being drawn.
        - Clear: clears
        - isEmpty: returns boolean indicating… well you know
    - Usage: Look at the btComm() method in SetupWizardUI.java for proper usage. It must have a ListView element in the layout file or else it fails
- CheckBoxItem.java

- Purpose: simple object that represents an entry in the CheckBoxAdapter
- Methods:
  - A bunch of getters and setters
- Usage: Used by Checkbox TextView

- CheckBoxTextView.java
  - Purpose: provide a UI wrapper for CheckBoxItem.java
  - Methods:
    - Constructor: does most the work in this class
    - setDevice: set the device to the specified one
    - setCheckBoxState: change the state of the checkbox
  - Usage: used to add entries to CheckBoxAdapter

com.anrl.micandroid.crypto

- AES.java
  - Purpose: provider an easy to use interface to AES encryption and decryption using the java.security libraries
  - Methods:
    - encryptByte: encrypt the passed byte array with the specified key using AES/CBC/PKCS5Padding
    - decryptByte: decrypt the passed byte array with the specified key using AES/CBC/PKCS5Padding
    - decryptByteNoPadding: decrypt the byte array using AES/CBC/NoPadding because the Ystring is too short to allow for padding
  - Usage: you can access directly, but most calls are done by DHKey.java's encrypt and decrypt methods
- Base64.java
  - Purpose: create base64 strings from byte arrays and create byte array from strings. We need this because AES ciphers operate over byte arrays which need to be transmitted. This implementation is credited to Robert Harder who has released it into the public domain
  - Methods:
    Note: These are just the ones that are used. The class provides many many more
    - Encode: encode a byte array as a base 64 string
    - Decode: decode a string into a bye array
  - Usage: Call Base64.encode and decode
- DHKey.java
  - Purpose: Provide a simple way to encrypt and decrypt data and do all of the work related to the DH key exchange
  - Methods:
    - initDH: this method takes the server's public key as input, generates the shared secret and returns our public key for the server to use
    - Encrypt: takes a string, converts it to a byte array, encrypts it with the shared key and then returns it as a base64 encoded string
    - Decrypt: takes a base 64 encoded string, turns it into a byte array, decrypts it using the shared key and returns the reconstructed plain text
  - Usage: Look in NetComm.java to see it implemented. Initialize it with the server's pubkey and the it's usable for encryption/decryption

- MD5hash.java
  - Purpose: provide an easy to use interface to java.security's implementation of an MD5hash
  - Methods:
    - Hash: take either one or two byte arrays and returns a md5 hash of them
  - Usage: Hash stuff, like the cloudkey and the nonce
- Random.java
  - Purpose: provide an easy way to generate a random byte array
  - Methods: generateRandom: does what it says using java.security
  - Usage: Generate random. Used in cloud formation to generate keys
- Registration.java
  - Purpose: To store and modify our devices own registration state. Probably not thread safe. Stores data using android's preference interface.
  - Methods:
    - getRegistration: registration is a singleton and this is the only way to acquire it. You need to pass it context because only with context can we access android's preference storage

        - Registration: attempt to load in stored preferences.
        - Register: tries to register, returns true if successful
        - Unregister: tries to unregister, returns true if successful
    - Usage: used to access the MICA variables that we need to respond to listreqs, hashreqs and register/unregister. Look at ListenService to see how it's used
- Secrets.java
    - Purpose: A general static helper class that contains methods for turning byte arrays into strings and vice versa, generate the hash message, obtaining the cloudkey and generating a random array
    - Methods:
        - getHashMessage: Returns a proper hash message
        - getCloudKey: Returns the cloud key
        - getY: returns a decrypted Y string based on the inputs
    - Usage: used to manipulate values in Registration into keys and secrets. Check out ListenService
- Util.java
    - Purpose: Utilities that don't fit anywhere else
    - Methods:
        - makeBytesfromString: Take a string and turn it into a 16 byte array
        - makeStringFromBytes: take a 16 byte array and turn it into a string
    - Usage: used in a ton of places.

com.anrl.micandroid.netcomm

- Auth.java
    - Purpose: Provide and implementation of the Authentication phase of MCIA. It is an object that requires minimal interaction, but if successful will return the username and password
    - Public Methods:
        - Constructor: initializes Netcomm and the rest of the fields we need
        - Authenticate: Start the thread that will
            1) Find all nearby devices in the cloud
            2) Contact the server and get the all the things we need to do a hash
            3) Send out a hash req
            4) Reply to the server with our hashes
            5) Deal with the server's response!
    - Private Methods:
        - sendDevices: this method uses the devices in the set that the ListReq method returns. It formats them all nicely, decides what to do with the server's reply containing devices to HashReq. It then sends out the hashreq. Starts a thread
          It's called once our handler has received a message from netcomm that we're encrypted
        - combineHashes: this method combines all the hashes we've received and sends it on to the server. It then takes the server's reply and sees if we've been authenticated. It is called from the handler after hashreq is done. Starts a thread
    - The Handler:
        - The handler does several things
            1) Deals with all bluetooth messages. It stores the devices that get returned and initiates the next part of the
            2) Deals with all the messages from netcomm about errors and succesful encryption setup.
    - Usage: Create a Auth object and call authenticate. It's used in MainUI.java
- Dissolve.java
    - TODO
- Formation.java
    - Purpose: This is mostly a worker thread for WizardUI.java . It does all the server communication and passes results back to a handler.
    - Public Methods:
        - doStep2: called after startConnectThread. Tries to login to the server. If it works, send a message to the handler
        - doStep3: Sends a list of a devices to the server. It takes the reply and figures out where to send the regreq messages to. It also generates the cloudkey and the upkey.
        - doStep4: encrypts the username and password and sends it to the server. It then cleans up and sends a message to the server
    - Private Methods:

        - formateDeviceList: returns the proper list of all the devices to send to the server
    - Usage: It is almost required to be used with a handler. See WizardUI for proper usage and stuff.
- NetComm.java
    - Purpose: Provider an extendable base for contacting the server. It contains read and write methods that will decrypt/encrypt. It will also setup and handle all of the encryption we do between the server and client.
    - Public Methods:
        - setHandler: set the handler that messages will be passed to
        - startConnectThread: launches a thread that opens the socket to the server, notifies the server of the type of connection we are making, and sets up an encrypted tunnel
        - endConnectThread: close the buffered readers/writers and the socket. Notify the handler we are done. Takes an parameter to decide whether or not to throw a general error back the handler
    - Protected Methods:
        - readIn: Takes input from the sockets, decrypts it if you pass it true, checks for headers, and passes back the string
        - writeOut: adds headers, encrypts if you pass it true, and then writes to the socket
    - Usage: Auth and Formation both extend NetComm so look at their usage to see how it's done

com.anrl.micandroid.UI

- MainUI.java
    - Purpose: The central UI. It is a dashboard at the moment, but another design could be used. Most of the UI stuff isn't super bound to the code
    - Public (Overridden) Methods:
        - onCreate: This method is called whenever the activity is launched. It handles all the UI drawing, and sets up the event listeners.
        - onCreateOptionsDialog: initializes the options menu when the menu button is pressed
        - onOptionsItemSelected: figure out which button got pressed and take the appropriate action
        - onCreateDialog: This creates a dialog for us to use when we are doing work in the background
    - Private Methods:
        - Test: launches the test activity which is mostly blank at this point
        - Wizard: launches the WizardUI activity unless you are registered, in which case it bails
        - MICAUtil: launches the MICA Utility activity
        - beginServicing: starts the local service
        - endServicing : ends the local service
        - Dissolve: not yet implemented, but will eventually launch the dissolve activity
        - Auth: creates an Auth object and starts the authentication process
        - makeToast: creates a small popup
    - Handler: This handler deals with messages from Auth. It's pretty simple compared to the one in WizardUI
- MICAUtil.java
    - Purpose: provide an interface to test a MICA device and see if it can respond to bluetooth messages properly. Contains code to interface with list, reg, unreg and hash
    - Public (Override) Methods:
        - onCreate: This method is called whenever the activity is launched. It handles all the UI drawing, and sets up the even listeners.
        - onCreateDialog: This creates a dialog for us to use when we are doing work in the background
    - Private Methods:
        - Listreq: Gets a list of previously paired devices and sends them to ListReq
        - Hashreq: uses the list of discovered devices from ListReq and sends them to HashReq along with a made up nonce/key
        - Unreg: Sends unregister requests with unreqreq to all the devices that we got listreqs from
        - Reg: WARNING: badly done. Registration requires sending a unique Y string to each mica device. Because of this RegReq only acts on one device at a time
    - Handler:
        - mHandler does different things depending on what kind of message it is
        - Hash:
            - Done: dismiss the loading dialog
            - Fail: add the device to the array adapter with a status: failed hash
            - Success: add the device to the array adapter with status: provided a good hash

            - Loading: clear the array adapter and show the working dialog
        - List:
            - Done: dismiss the loading dialog
            - Fail: don't do anything
            - Success: add the device to the array adapter with it's address and add it to the set of devices we have found
            - Loading: clear the set of found devices and the array adapter and show the loading dialog
        - Reg:
            - Success: add the device to the array adapter with a status: register success
            - Fail: add the device to the array adapter with a status: register fail
        - Unreg:
            - Done: dismiss the dialog
            - Fail: Add the device to the array adapter with a status of unregister fail
            - Success: add the device to the array adapter with a status of unregister succes
            - Loading: clear the array and display the loading dialog
- SetupWizardUI.java
    - Purpose: Provides an interface to create a MICA cloud. Sends hashreqs, get secrets, etc…
    - Public (Overridden) methods:
        - onCreate: this method attempts to restore itself to a previous state if possible. It does that by looking for extras in the bundle it is passed and on getLastNonConfigurationInstance. It also runs setScreen which figures out which screen to be displayed
        - onKeyDown: This is called when any key is pressed. We only do something if it's a back key.
        - onSaveInstanceState: This just puts a screen_state into the bundle which gets passed to onCreate
        - onRetainNonConfigurationInstance: This is where we create the object we receive when we call getLastNonConfigurationInstance. We return the serverThread and if applicable the contents of the listreq
        - onCreateDialog: This draws our loading screen
    - Private methods:
        - Discover: clears the checkbox adapter and begins a listreq
        - Begin1: This method sets up the first screen layout and all of the even listeners it needs. It also begins the first step of the serverThread
        - Login2: This method sends the username/login stuff to server via Formation.step2
        - Bluetooth3: Sets up the bluetooth step. It calls discover via a button. It also contains all the code that is needed to initialise the checkbox list adapter
        - Final4: Sends the U/P combo to the server and finishes the thread
        - Success: Shows the user they have successfully setup a cloud
        - Error: shows the error screen, shows an error message and closes the thread
        - Warn: warns is called when back is pressed. It shows a yes/no dialog that prevents the user from accidentally hitting back
        - Cancel: closes the server thread and finishes the activity
        - setScreen: takes a parameter to choose which screen to display.