Java API for Unifying Ad-Hoc Wi-Fi Networking

Instruction Manual

Team Members

Peter Banis - pbanis2015@my.fit.edu
Klaus Çipi - kcipi2015@my.fit.edu
Michael Kolar - mkolar2015@my.fit.edu
Robert Olsen - olsenr2015@my.fit.edu

Faculty Sponsor

Dr. Marius Silaghi - msilaghi@fit.edu

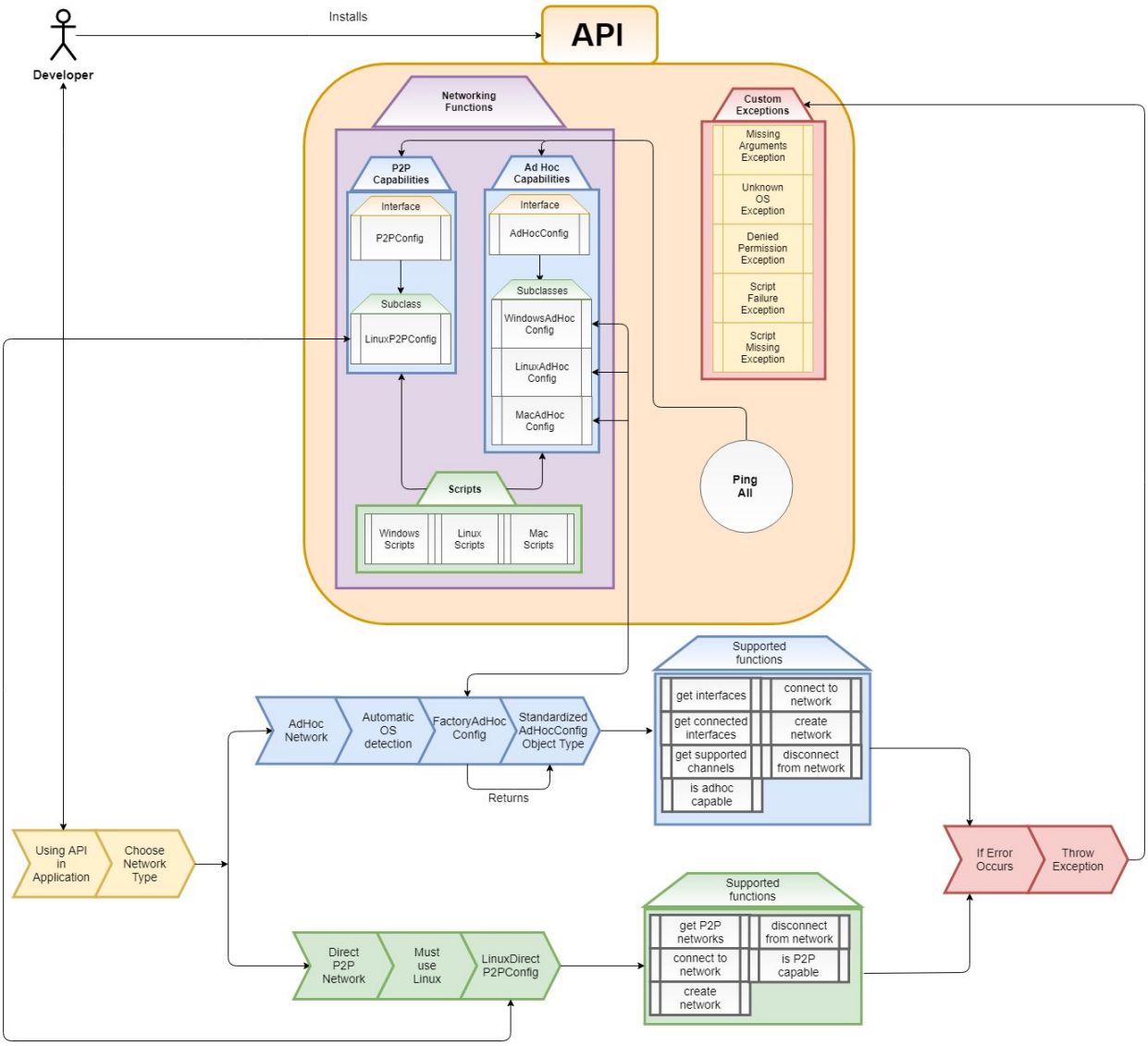
Table of Contents

Introduction

The Java API for Unifying Ad-Hoc WiFi Networking is a toolset to assist software developers in creating applications which depend on cross-platform networks. The API is constructed as a series of scripts that perform operating system specific commands wrapped in Java classes. At the core of the API is the abstract configuration class, which helps to generalize the networking processes common between Windows, Mac, and Linux devices. Each operating system extends these abstractions and implements them using their system-specific procedures and scripts. We are able to accomplish this due to the portable nature of the Java language, which natively provides the tools needed to determine the operating system the API is running on and the presence of the JVM on all systems. Once the operating system is identified, we are able to create a configuration object that is catered to that architecture. The power of this API is that it is one library that extends its reach across multiple different operating systems.

Inside this manual, you will find how the API handles Ad-Hoc networks, Wi-Fi Direct networks, and error reporting. The functions for network interaction are provided in a generalized form, and any platform specific differences for a given function may be found in the “Operating System Differences” section. What you will NOT find is how to build an application off of the API (sharing internet access, file transfer, etc), as that is beyond the scope of the project. WiFi-Direct support for Android was planned, and while development on it was started, the overall inexperience of the team with Android proved to be too great of a challenge.

System Design Diagram



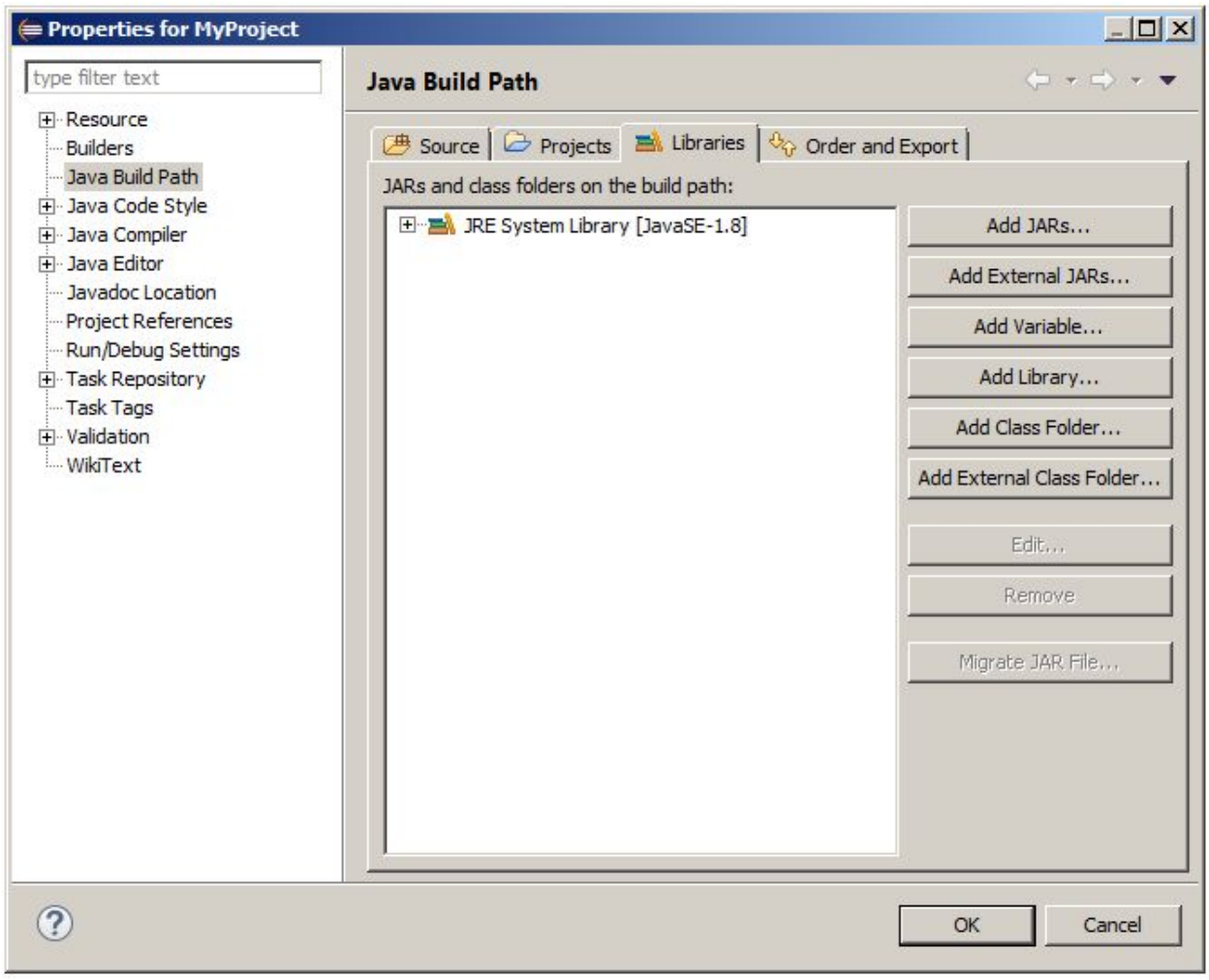
Installation Procedure

1. Download the JAR file from our website: http://adhocapi.com

Eclipse

2. Right click on your project folder and select “Properties”

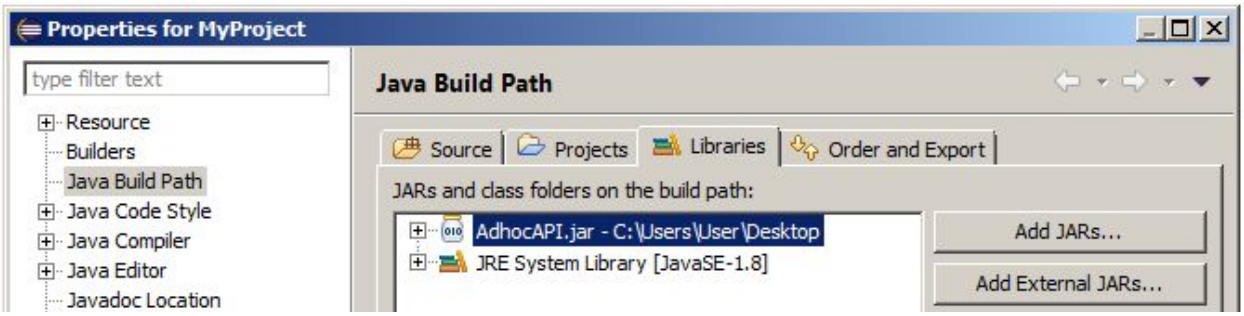
3. Navigate to the “Libraries” tab under “Java Build Path”



4. Click on “Add External JARs...”

5. Browse for the JAR file, select it, and hit OK

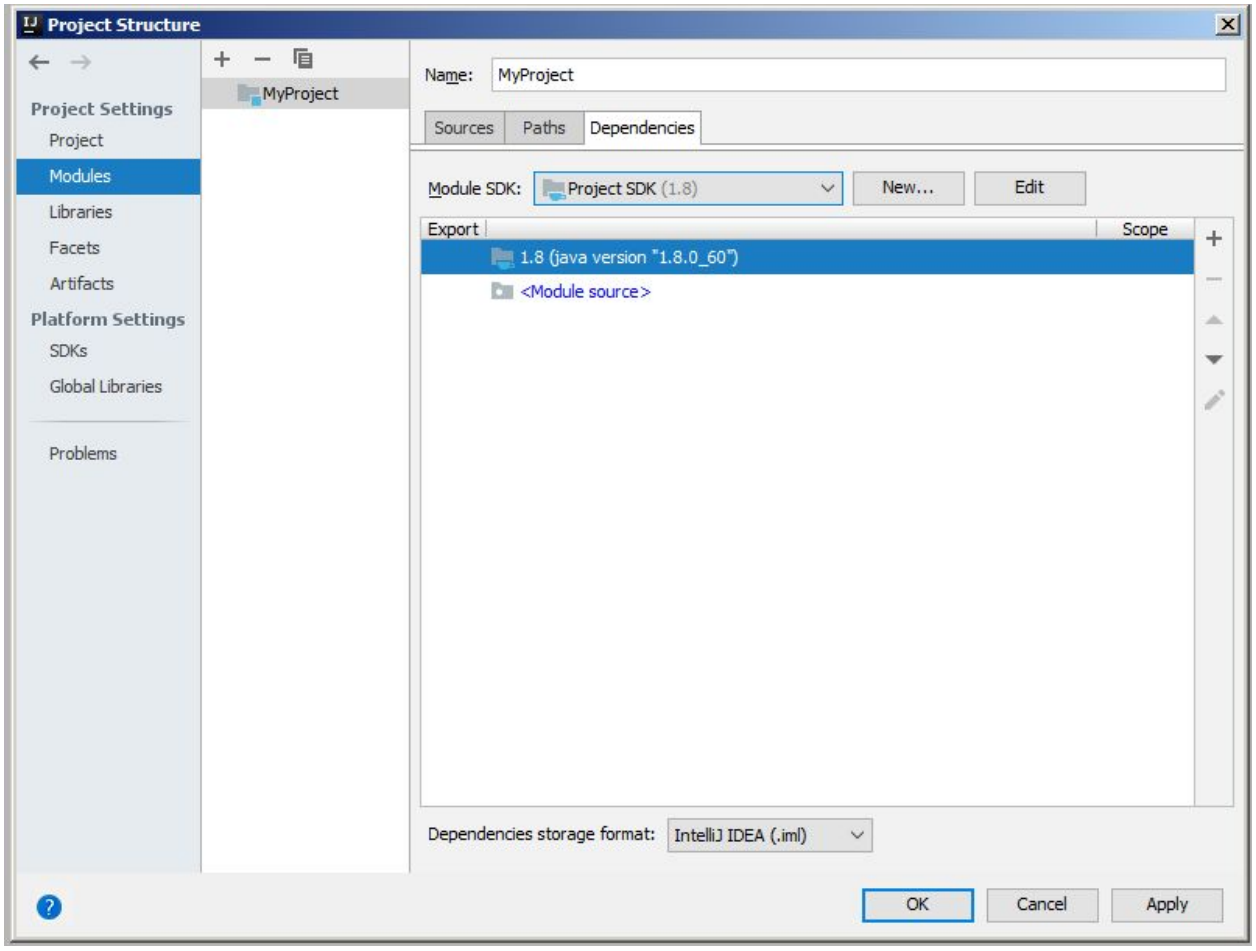
6. Once the JAR appears in your “Libraries” tab, hit OK



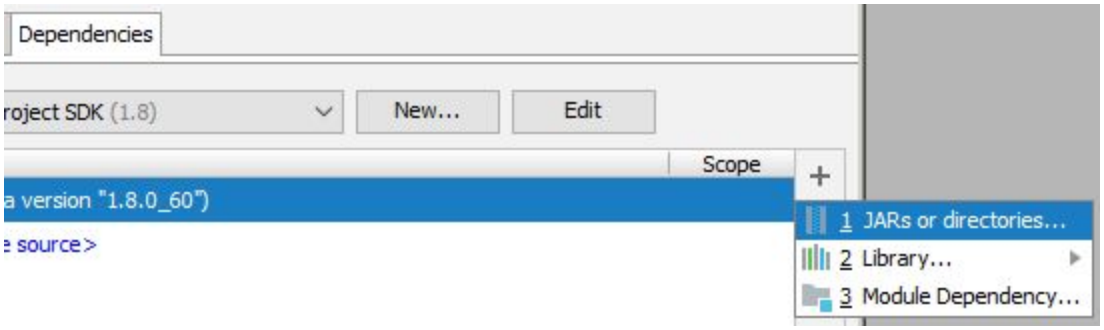
IntelliJ

2. Under “File” on the toolbar, select “Project Structure”

3. Navigate to the “Dependencies” tab under “Modules”

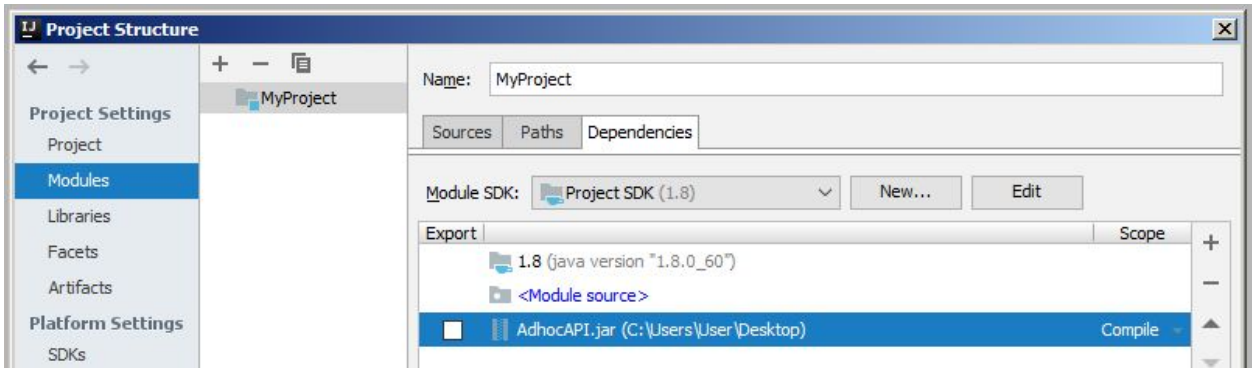


4. Click on the + on the side and select “JARs or directories...”



5. Browse for the JAR file, select it, hit OK

6. Once the JAR appears in your “Dependencies” tab, hit Apply and then hit OK



API Overview

Ad-Hoc Functions

public int getChannel()

Get the current value of the channel field for the current configuration.
Returns: Channel number of the configured network

public String getPassword()

Get the current value of the password field for the current configuration.
Returns: Password of the configured network

public String getSSID()

Get the current value of the SSID field for the current configuration.
Returns: Name of the configured network

public String getNetworkInterface()

Get the current value of the networkInterface field for the current configuration.
Returns: Name of the interface that is being used to connect to or host the configured network

public String[] getInterfaces()
throws ScriptFailureException, MissingArgumentsException, ScriptMissingException, DeniedPermissionException

Sends a request to the operating system to inform the API of all known network interfaces.
Returns: A list of the names of all network interfaces

public String[] getConnectedInterfaces()
throws ScriptFailureException, MissingArgumentsException, ScriptMissingException, DeniedPermissionException

Sends a request to the operating system to inform the API of all known active network interfaces.
Returns: A list of the names of all enabled and connected network interfaces

public int[] getSupportedChannels(String wirelessInterface)
throws ScriptFailureException, MissingArgumentsException, ScriptMissingException, DeniedPermissionException

- String wirelessInterface: The network interface to check for channels

Sends a request to the operating system to inform the API of the range of channels that are supported for a given network interface.
Returns: A list of all the supported channels on a network interface

public int connectToNetwork()
throws ScriptFailureException, MissingArgumentsException, ScriptMissingException, DeniedPermissionException

The API will attempt to create a network using the current fields in the configuration. If there is already a network with the same settings, they will merge together. The API will then attempt to join to the network with the matching settings.
Returns: Zero on success

public int connectToNetwork(String networkName, String password, String interfaceName, int channel)
throws ScriptFailureException, MissingArgumentsException, ScriptMissingException, DeniedPermissionException

- String networkName: The name of the network to join
- String password: The password of the network to join
- String interfaceName: The name of the network interface to join the network with
- int channel: The channel the network interface will use to join the network with

The API will attempt to create a network using the parameters provided in the function call. If there is already a network with the same settings, they will merge together. The API will then attempt to join the network with the matching settings.
Returns: Zero on success

public int createNetwork()

throws ScriptFailureException, MissingArgumentsException,
ScriptMissingException, DeniedPermissionException

The API will attempt to create a network using the current fields in the configuration. If there is already a network with the same settings, they will merge together.
Returns: Zero on success

public int createNetwork(String networkName, String password, String interfaceName, int channel)
throws ScriptFailureException, MissingArgumentsException,
ScriptMissingException, DeniedPermissionException

- String networkName: The name of the network to join
- String password: The password of the network to join
- String interfaceName: The name of the network interface to join the network with
- int channel: The channel the network interface will use to join the network with

The API will attempt to create a network using the parameters provided in the function call. If there is already a network with the same settings, they will merge together.
Returns: Zero on success

public int disconnectFromNetwork()
throws ScriptFailureException, MissingArgumentsException,
ScriptMissingException, DeniedPermissionException

The device will attempt to disconnect from the current network (if there is one). It is recommended that you use this function instead of manually disconnecting from a network, as it may perform additional cleanup.
Returns: Zero on success

public Socket clientSocket(String ip, int port)
throws IOException

- String ip: The IP address for the client socket
- String port: The port for the client socket

Creates a new Socket object, bound with the specified ip address and port.
Returns: The resulting Socket

public ServerSocket serverSocket(int port)
throws IOException

- int port: The port for the server socket

Creates a new ServerSocket object, bound with the specified port.
Returns: The resulting ServerSocket

public boolean isAdhocCapable()

Evaluates whether or not the device is capable of hosting and joining Ad-Hoc networks.
Returns: True if Ad-Hoc capable, False if not

public static List<String> ping(String ipDomain)

- String ipDomain: An IP Address where the first 24 bits are dedicated to the network identifier and the last 8 bits are dedicated to the host identifier (Format: X.Y.Z.0)

Scans the network for connected devices. This is accomplished by pinging all possible hosts whose network identifiers match against the network identifier of the domain, ranging from X.Y.Z.1 to X.Y.Z.254. The results of the iterative pinging are stored in a list, and the IP addresses of the responsive devices are picked out of the results list and stored in the new list of connected devices. The IP addresses of the devices on a network must be static or they will not be discovered.
Returns: A list of IP addresses of the connected devices on a network

public void setChannel(int c)

- int c: The new network channel

Sets the channel field in the current configuration to its new value.

public void setSSID(String ssid)

- String ssid: The new network name

Sets the SSID field in the current configuration to its new value.

public void setPassword(String password)

- String password: The new network password

Sets the password field in the current configuration to its new value.

public void setNetworkInterface(String networkInterface)

- String networkInterface: The new network interface

Sets the networkInterface field in the current configuration to its new value.

Wi-Fi Direct Functions

public void StringMACPair[] getP2PNetworks()

Scans surrounding area for active Wifi-Direct networks. Each network has the name of the network and MAC address of the group owner.
Returns: An array containing network names and corresponding MAC addresses.

public int connectToNetwork(String MAC)

- String MAC: The MAC Address of the group owner

Connects to the P2P group using the Push Button method.
Returns: Zero on success, One on failure

public int connectToNetwork(String MAC, String PIN)

- String MAC: The MAC address of the group owner
- String PIN: The PIN is the PIN of the network to use in connecting

Connects to the P2P group using the PIN method.
Returns: Zero on success, One on failure

public int createNetwork(String networkName)

- String networkName: The suffix to be added to the P2P network.

Creates a p2p-interface if needed, and then creates a new p2p group with this device as group owner.
Returns: Zero on success, One on failure

public boolean isP2PCapable()

Evaluates whether or not the device is capable of hosting and joining P2P networks.
Returns: True if P2P capable, False if not

public int disconnectFromNetwork()
throws ScriptFailureException, ScriptMissingException, DeniedPermissionException

Disconnects from the current P2P Group, removing the p2p-interface that was created if needed.
Returns: Zero on success, One on failure

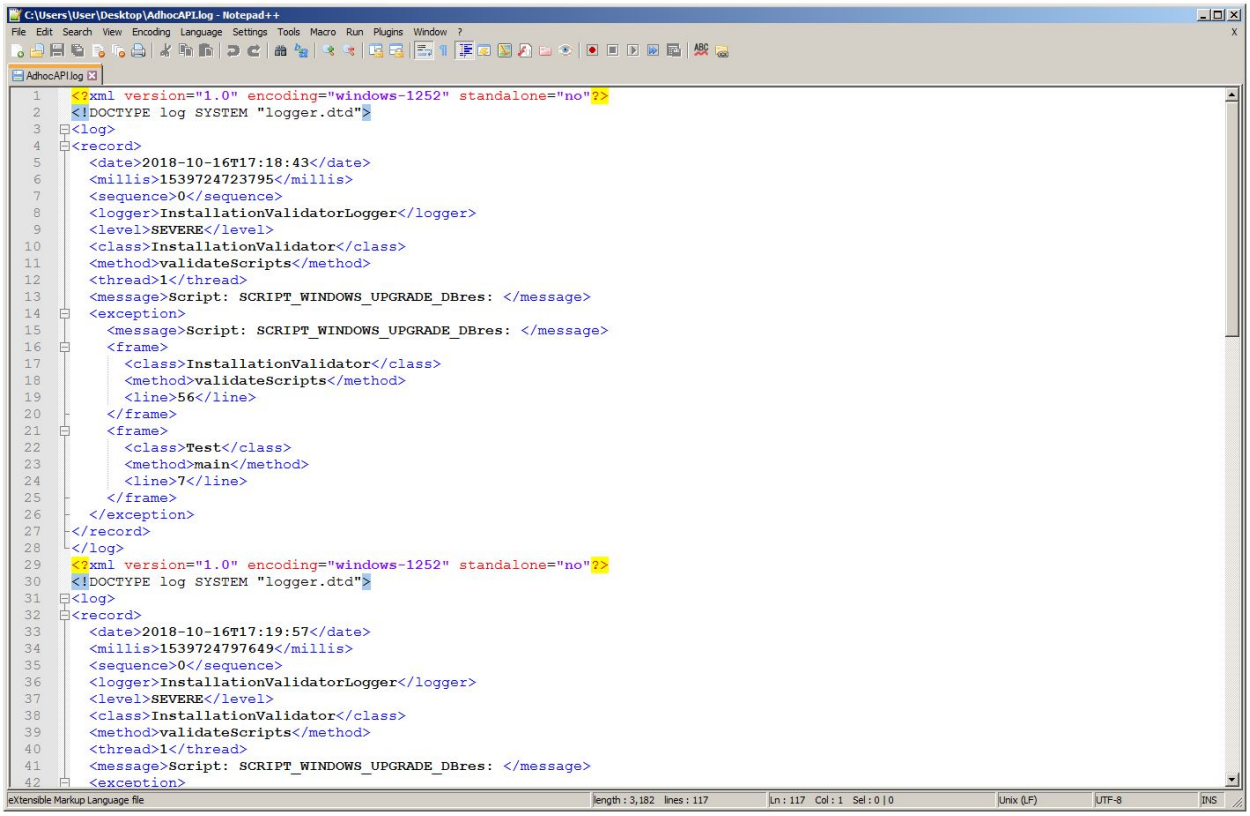
Error Reporting

In addition to the standard Java exceptions, the API has defined its own unique set of exceptions to help give more specific feedback when things don't go quite right. Since the error reporting processes extrapolate information from the exceptions that are raised while using the API, it's important to know the purpose of these custom exceptions.

- DeniedPermissionException: Occurs when the API is denied permission to access (execute) a file.
- MissingArgumentsException: Occurs when the fields of a configuration are unset and a function that retrieves and uses them is called.
- ScriptFailureException: Occurs when a script is successfully executed but doesn't terminate properly (Non-zero exit code).
- ScriptMissingException: Occurs when a script that the API depends on is not properly named or not in its proper location.
- UnknownOSException: Occurs when the API does not recognize the operating system. The API expects the operating system to contain "windows", "linux", or "mac os" in its name.

One method the API uses to report errors is the logger. Reports will be stored in the "AdhocAPI.log" file. The severity of an error will be logged as either "SEVERE" or

“WARNING”, depending on where the problem occurs in a function. If the function can attempt to provide the same outcome from a different perspective, it will be treated as a warning. But if there is no way for the function to complete successfully, then it will be treated as a severe issue. The logger will also display a message via the toString() function if the error is an exception, or a hardcoded string otherwise (usually the case for warnings). Exceptions with a level of severe will also provide a stack trace in the log file. The log file will also contain other smaller pieces of information, such as the time, date, and thread that the error occurred on.



Example log file

In the prior two sections, several of the functions claim to return zero upon successful completion. The Wi-Fi Direct functions will otherwise return one, but what happens when the Ad-Hoc functions don’t succeed? What values do they return if they do manage to fail? Since failures in function execution are marked by exceptions, there is no return value. The API prints a stack trace to the console and attempts to continue execution without the information from the function, if possible.

Operating System Differences

Linux

- If a password for the network is set or passed as an argument, the program must previously have called getSupportedChannels(). This is due to a particular format requirement of configuration files for wpa_supplicant. Thus, we recommend calling getSupportedChannels() before any call to create or join a network when passwords are used.
- The API will automatically set your IP to a static address when you connect to a network, which helps devices find each other on the network.

Mac OS

- The getConnectedInterfaces() function is not supported on Macs.

Windows

- connectToNetwork(String networkName, String password, String interfaceName, int channel) and createNetwork(String networkName, String password, String interfaceName, int channel) will both update the current configuration's fields. This does not happen with the equivalent functions on Linux or Mac OS.
- The createNetwork and connectNetwork functions on Windows are virtually the same, due to how profiles work. Windows requires a network profile regardless of whether you intend to make or join an Ad-Hoc network, since this is how Windows identifies what the network it is connecting to is or what the network it should be opening will be. Once a new profile is created, the corresponding network will populate the list of possible connections. If you join this Ad-Hoc network before it is publicly visible, the network will be hosted. If you join the network while it is publicly visible, you will connect to that visible network.
- As stated prior, every Ad-Hoc network you interact with on Windows requires the creation of a network profile. The first step of creating a profile is to generate an XML file, which is mostly just a generic skeleton that requires the SSID (and its hexadecimal representation) of the network you're attempting to create or join. The profiles you create are not cleaned up until disconnectFromNetwork() is called. If you neglect to call this function, you will have to manually delete the profiles yourself. Regardless of whether or not you use the aforementioned function, you will still have to delete the XML files manually.

- The API will automatically set your IP to a static address when you connect to a network, which helps devices find each other on the network. It will also reset this by re-enabling dynamic addresses when disconnectFromNetwork() is called.
- Windows may appear to be on a network, even though it has yet to fully establish a connection with it. This is just a strange quirk of the operating system, and is not a result of any overhead in the API. During this period, you may not be able to see or communicate with other devices on the network, and vice-versa. Please remain patient until Windows is able to tag the network as public, work, or private.

Examples of Users

User categories: *API Providers, Software Developers, End-users*

API Providers:

These are the people who bear the burden of running maintenance on this API as legacy code. These people help design new features, implement those features, test them, and distribute the API to software developers. API providers have an obligation to update the API in order to make it viable as technology changes; this is especially relevant because new versions of Windows, macOS, Linux distributions, and even popular new operating systems altogether will be created in the future. The API handles the “under-the-hood” implementations and networking issues unique to each operating system, so this requires attention and potential re-imaginings of new functionality in order to accommodate future developments.

Furthermore, API Providers need to ensure that the API is appealing to software developers as it allows them to create applications which pleases end users. This is essential in order to help prevent the API from fading into obscurity in the modern programmer’s awareness; this is especially relevant as competitors emerge which seek to steal from the API’s user base or to even replace it altogether.

The API Providers also need to make the API distribution as seamless as possible. Typically, they set up a portal or system which showcases the API and to communicate with potential users. One common way this is accomplished is by setting up a website where prospective software developers can visit to overview documentation which explains the API’s purpose as well as other background information and the software developers can download the API.

API Providers shall pursue some further aspects aside from the API’s up-to-dateness that will ensure the API is appealing. The API shall be modular. Modularity helps keep the API from becoming too convoluted and too bloated. This requires dilliquence, but it’s mutual beneficial for both the software developers who are API users and the software developers who act as API providers and are responsible for future maintenance. The API shall also be efficient. The applications based on the API can only be as quick and memory efficient as the API permits them to be. Extensive effort was put into solving any occuring problems in worst case linear time complexity, so future API Providers have a heavy torch to carry. Finally, the API shall be well documented. This includes both inside the code itself for maintenance purposes as well as on the website to enable transparency for potential customer software developers.

Software Developers:

These are the people who are consumers of the API so that they can use it to create their own applications. Due to the nature of this API it only needs to be imported into applications designed to handle Ad-Hoc networks. They benefit from the API because the API reduces the amount of code they need to write and test which makes programming much less time consuming and stressful. Software Developers are the core consumer base for the API since they are the only party which directly uses the API.

They are the core stakeholder, so their needs must be addressed in order to keep the API in use. First, the API is general enough such that the software developers aren't locked into design decisions which they don't wish to follow. For example, the API doesn't have an in-built encryption algorithm, because that would limit its flexibility. Instead the software developers can choose whether to include encrypt and what encryption algorithm is most appropriate for their purposes. Additionally, the software developers will want well-written code; this means that the code actually works, any implementations can work in a reasonable time complexity, the code is concise and easy to read. They also want proper documentation to be available to them which explain what the classes are called, the names of the methods, what the methods do, and any details necessary to understand the API's overall structure and capabilities. Finally, they will want the API to be easy to understand; after all, if the API is a burden to use, then they may as well program their applications' underlying networking functionality by themselves or by using a competitor's API.

End-users:

These are the people who use the API indirectly through applications and thus often don't realise that they are users at all; these applications are build by the software developers who are consumers of the API. Essentially, this means that there are various parties interested to the API who affect each other down a pipeline and the end-users are the final party at the end of the pipeline. Appealing end-users is the reason that the code development pipeline was constructed to begin with. They are important because their desires are reflected in the applications which are designed for them. The features of those applications must also in turn be reflected in the APIs that those applications draw from. This means that changes in the End-users may result in necessarily changes in the API in order to accommodate those new needs.