AQL-System: Manual

(https://github.com/FoelliX/AQL-System/wiki)

Table of contents

Menu

AQL

The Android App Analysis Query Language (AQL)

The AQL consists of two main parts, namely AQL-Queries (compositions of AQL-Questions) and AQL-Answers. AQL-Queries enable us to ask for Android specific analysis subjects in a general, tool-independent way.

Tutorials

Questions
Answers

Links

Question-Grammar
Answers-Schema
FAQ

Questions

AQL-Questions & -Queries

AQL-Questions can be used to ask various analysis tools for certain analysis subjects in a general way. AQL-Queries represent compositions of AQL-Questions combined by AQL-Operators.

Structure

All possible AQL-Queries are defined by this grammar. In general, a query consists of one or more questions possibly connected by AQL-Operators. A question in turn exist of an analysis subject and a target description.

Analysis Subject

Each question can ask for one of the following analysis subjects:

- `Flows`: A flow symbolizes the transfer of information from one program location to another. To describe such a flow, these two locations have to be specified
- `Intents`: Intents are used for inter-component communication, where one component sends an intent to another component. In case of an explicit intent the receiver can be identified by a component reference. But in case of an implicit intent the receiver needs to be recognized through the information triple action, category and data
- `IntentFilters`: Specifies the types of intents that an activity, service, or broadcast receiver can respond to
- `IntentSinks`: Intent-sinks are special intents, so they can be described in the same way as intents can be described. However, in this case a reference to a program location is always required *(Required to connect answers -- See `CONNECT` operator)*
- `IntentSources`: Intent-sources represent the counterpart of intent-sinks. An intent-source might be the starting location of an information flow path. They can be described just like intent-sinks and intents with one small but important difference: The reference has to refer to a statement that, for instance, extracts information from an intent. *(Also required to connect answers -- See `CONNECT` operator)*
- `Permissions`: Shows which permissions are used by a reference

Analysis Target

The analysis target is specified by a chain consisting of the following elements
`Statement -> Method -> Class -> App`
In such a chain the elements `Statement`, `Method` and `Class` are optional. This allows to ask for on-demand properties for certains parts of an app. In a complete question we can ask for information inside one target (`IN`) or for information between two targets (`FROM` ... `TO`). Furthermore, a certain preprocessor can be assigned to be apllied before an app is analyzed.

Operators

Any question contained in a query ends with a `?`-symbol. This indirectly refers to the answer which is received by asking this question. It is also possible to directly reference a previously computed answer. Therefore an `!` at the end distinguishes directly addressed answers from indirectly addressed ones.

The following operators can be used to filter or combine answers.

- `FILTER`: Outputs the input set, but beforehand it removes all permissions, intent-sinks and -sources whose reference does not appear in any flow contained in the answer. Intents and intent-filters from the input set are kept in the output set. The filter operator can also be used together with an analysis subject in order to filter out all elements of the selected subject of interest or with a name-value-pair to filter elements that contain this name-value-pair as attribute.
- `UNIFY`: Collects all information from two different AQL-Answers and puts it into one.
- `CONNECT`: Works as `UNIFY`, however, it additionally computes transitive flows and flows that can be determined by connecting intent-sinks with intent-sources.

To identify boundaries of operators, `[` and `]` are used.

It is also possible to define your own operator (see the configuration tutorial).

Examples

Question Examples

The following question asks for flows inside app *A*:

```
Flows IN App('A.apk') ?
```

The next one for flows between app *A* and *B*:

```
Flows FROM App('A.apk') TO App('B.apk') ?
```

To ask for the permission(s) used by a specific statement inside app *A*, the following question can be constructed:

```
Permissions IN Statement(sendTextMessage(..))->App('A.apk') ?
```

Let us assume we got a preprocessor associated with the keyword *TEST*. To ask for Intents in a preprocessed version of *A* we formulate:

```
Intents IN App('A.apk' | 'TEST') ?
```

To influence the tool selection a specific tool can be choosen:

```
Flows IN App('A.apk') USES 'AwesomeDroid' ?
```

Or the tool with the highest priority for a certain set of features can be selected:

```
Flows IN App('A.apk') FEATURING 'TEST', 'Awesome' ?
```

Query Examples

Let us assume we want to know which permission protected statements are connected. The question we could ask is:

```
UNIFY [
    Flows IN App('A.apk') ?,
    Permissions IN App('A.apk') ?
]
```

Assuming we downloaded an answer telling us which permission uses can be found in app *A* we could use the following query:

```
UNIFY [
    Flows IN App('A.apk') ?,
    'downloaded_permission_answer.xml' !
]
```

We could further filter this result by adding the `FILTER` operator. In this case we would only get Permissions that are somehow related to a flow:

```
FILTER [
    UNIFY [
        Flows IN App('A.apk') ?,
        Permissions IN App('A.apk') ?
    ]
]
```

To connect two answers determined for two different apps (*A*, *B*) the `CONNECT` operator can be used. Thereby, flows are generated for each pair of matching intent-sinks and -sources:

```
CONNECT [
    IntentSinks IN App('A.apk') ?,
    IntentSources IN App('B.apk') ?
]
```

The `CONNECT` operator can also be used to compute the transitive closure of a set of flows:

```
CONNECT [
    Flows IN App('A.apk') ?
]
```

Transformation Example

The following queries may lead to the same answer:

```
Flows FROM App('A.apk') TO App('B.apk') ?
```

```
FILTER [
    CONNECT [
        Flows IN App('A.apk')?,
        Flows IN App('B.apk')?,
        CONNECT [
            IntentSinks IN App('A.apk') ?,
            IntentSources IN App('B.apk') ?
        ]
    ]
]
```

(The AQL-System automatically applies such transformations if required, e.g. because of missing appropiate tools to answer the initial query.)

Answers

AQL-Answers

AQL-Answers are used to represent results of analysis tools in a general and precise way. The following analysis information can currently be represented through AQL-Answers:

- Flows (e.g. taint flows)
- Intents
- Intent-Filter
- Intent-Sinks
- Intent-Sources
- Permissions

The structure of AQL-Answers is precisely defined through an XML Schema Definition. Consequently, every answer is represented by an .xml document. Such an .xml document is structured as follows:

```
<answer>
    <flows>
        <flow>
            ...
        </flow>
        ...
    </flows>
    <intentsources>
        <intentsource>
            ...
        </intentsource>
        ...
    </intentsources>
    ...
</answer>
```

Each `flow`, `intent`, `intentfilter`, `intentsink`, `intentsource` and `permission` element can hold arbitrary many `attribute` elements consisting of name-value-pairs to represent additional information as shown in the following.

Example 1: Flows

One taint flow detected by a tool such as FlowDroid can be represented in the following way:

```xml
...
<flow>
    <!-- Flow starts from -->
    <reference type="from">
        <statement>
            <statementfull>$r4 = virtualinvoke $r3.&lt;android.telephony.TelephonyManager: java.lang.String getSimSerialNumber()&gt;()</statementfull>
            <statementgeneric>android.telephony.TelephonyManager: java.lang.String getSimSerialNumber()</statementgeneric>
        </statement>
        <method>&lt;de.foellix.aql.aqlbench.api19.interappstart1.MainActivity: void source()&gt;</method>
        <classname>de.foellix.aql.aqlbench.api19.interappstart1.MainActivity</classname>
        <app>
            <file>/media/sf_share/fix/InterAppStart1.apk</file>
            <hashes>
                <hash type="MD5">2aafeb4bd6e436f66fc06083fda3beb1</hash>
                <hash type="SHA-1">a4619c4448047436e96ec4397dff343e6702c532</hash>
                <hash type="SHA-256">627af4963cbd3b31a0e9c3ef4a029cfb62534689c2dd620c97252ba55c72ac15</hash>
            </hashes>
        </app>
    </reference>

    <!-- Flow ends in -->
    <reference type="to">
        <statement>
            <statementfull>virtualinvoke $r3.&lt;android.telephony.SmsManager: void sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.PendingIntent)&gt;("123456789", null, $r2, null, null)</statementfull>
            <statementgeneric>android.telephony.SmsManager: void sendTextMessage(java.lang.String,java.lang.String,java.lang.String,android.app.PendingIntent,android.app.PendingIntent)</statementgeneric>
            <parameters>
                <parameter>
                    <type>java.lang.String</type>
                    <value>"123456789"</value>
                </parameter>
                <parameter>
                    <type>java.lang.String</type>
                    <value>null</value>
                </parameter>
                <parameter>
                    <type>java.lang.String</type>
                    <value>$r2</value>
                </parameter>
                <parameter>
                    <type>android.app.PendingIntent</type>
                    <value>null</value>
                </parameter>
                <parameter>
                    <type>android.app.PendingIntent</type>
                    <value>null</value>
                </parameter>
            </parameters>
        </statement>
        <method>&lt;de.foellix.aql.aqlbench.api19.interappend1.MainActivity: void sink()&gt;</method>
        <classname>de.foellix.aql.aqlbench.api19.interappend1.MainActivity</classname>
        <app>
            <file>/media/sf_share/fix/InterAppEnd1.apk</file>
            <hashes>
                <hash type="MD5">fc850d773145bbc65694c58213f6cb6f</hash>
                <hash type="SHA-1">edd9bf89ef2b01be8abf7f6616dd5867c96a79ac</hash>
                <hash type="SHA-256">79dd06fa366c4313f84913b73ce8fe157dd6dab8b0946d1a54ef60b604f2a26d</hash>
            </hashes>
        </app>
    </reference>

    <!-- Attribute showing this is a complete flow -->
    <attributes>
        <attribute>
            <name>complete</name>
            <value>true</value>
        </attribute>
    </attributes>
</flow>
...
```

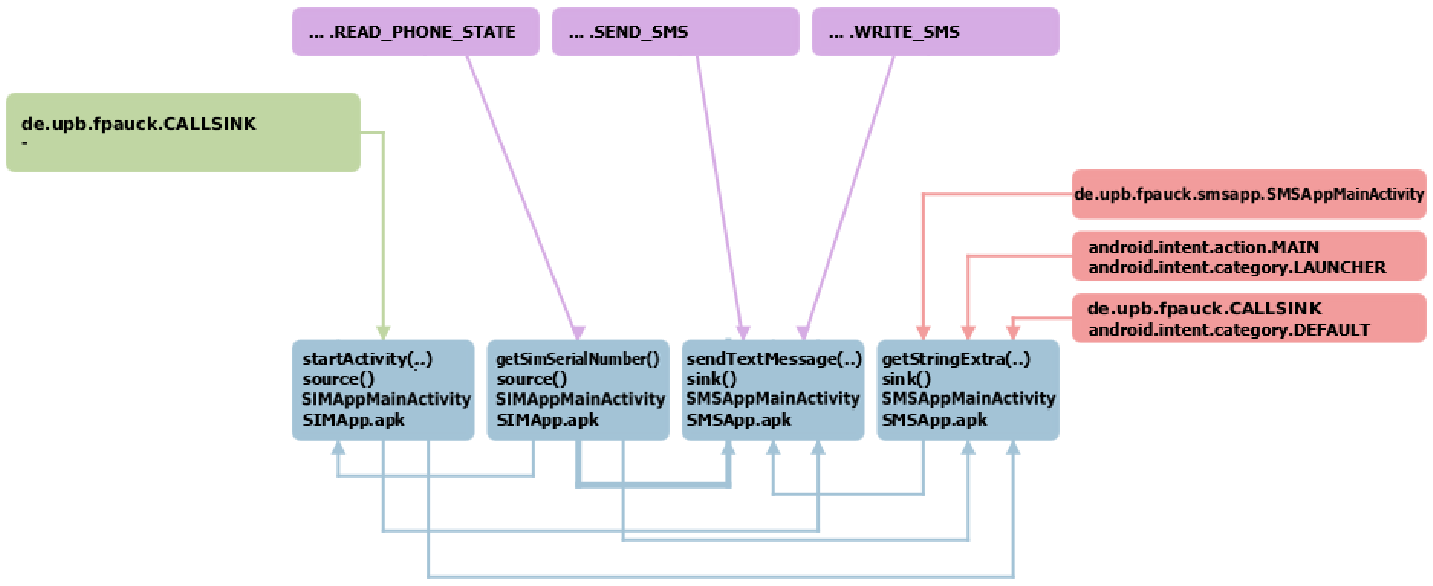
The statement where the `flow` starts is described by a `reference` -- Reference elements are used to describe certain program locations inside an app. This reference's `type` attribute is set to *from* to reflect that the flow starts at this program location. The program location is described by four inner elements, namely `statement`, `method`, `classname` and `app`. For any statement two String representations can be provided. In this example the `statementfull` element refers to the full Jimple statement, whereas the `statementgeneric` element only refers to the program independent part of it. Additionally, each statement has a list of name-value-pairs attached to it in order to represent the parameters of e.g. a function call. `method` refers to the signature of the method which holds the previously described statement. `classname` again refers to the class containing the beforehand described method. Lastly, the `app` element references the app where this class can be found. To do so, the app's *.apk* file can be specified along with arbitrary hashes to recognize the app on other systems.

The second `reference` element's `type` attribute is set to "to", thereby we know that the flow ends in this program location. Its structure is the same. Finally, the flow is fully described.

The *attributes* list at the end of the flow description holds one `attribute`. The name-value-pair (*complete* = *true*) representing this attribute refers to the fact, that this flow is complete -- It leads from a tainted source to a sink.

Example 2: Connected answer

The following figure shows an AQL-Answer in a graphical way. Such graphs can be generated with the AQL-System on the basis of an AQL-Answer .xml file.



The blue nodes and edges on the bottom represent the different flows. The green and red ones represent the intentsinks and -sources. Lastly, the purple on top represent permissions.
To produce this answer three different analysis tools where asked by means of the AQL:

- FlowDroid for Flows inside the SIMApp and SMSApp
  Thereby, among others a flow
  - from `getSimSerialNumber()` to `startActivity(..)` and a flow
  - from `getStringExtra(..)` to `sendTestMessage(..)` could be found.
    In particular, the complete flow
  - from `getSimSerialNumber()` to `sendTextMessage(..)` could **not** be found, yet!
- IC3 for IntentSoures and IntentSinks in these apps
  Its result especially holds one IntentSink and one IntentSource described by the action string *de.upb.fpauck.CALLSINK*.
- and PAndA² for Permission uses. The result shows that, on the one hand, the *READ_PHONE_STATE* permission is required by the `getSimSerialNumber()` statement and, on the other hand, the `sendTextMessage(..)` statement required the *SEND_SMS* and *WRITE_SMS* permission.

The AQL-Operator `CONNECT` has to be applied to connect the answers. In doing so, the IntentSink is matched to the IntentSource, resulting in a new flow from `startActivity(..)` to `getStringExtra(..)` which is the missing piece to finish the puzzle. Now the three flows found can be used to construct the complete flow (from `getSimSerialNumber()` to `sendTextMessage(..)`) in a transitive manner.

AQL-System

The Android App Analysis Query Language - System (AQL-System)

The AQL consists of two main parts, namely AQL-Queries (compositions of AQL-Questions) and AQL-Answers. The AQL-System takes AQL-Queries as input and outputs AQL-Answers.

New Tutorials

Along with the release of version 1.2.0 configurations must be upgraded:
Configuration Upgrades

Basic Tutorials

Runthrough
Install & Compile & Develop
Launch parameters
Configuration
Query execution
Add your tools

FAQ

Runthrough

Runthrough

The following instructions deal with the installation of the AQL-System. Along with that Amandroid will be installed. Hence, the AQL-System will be setup to use Amandroid only. (The operating system considered is Linux.)

1. Download the latest version of the AQL-System: here
   - Unzip it!
2. Download Amandroid: https://bintray.com/arguslab/maven/argus-saf/3.1.2
   (direct link: https://bintray.com/arguslab/maven/download_file?file_path=com%2Fgithub%2Farguslab%2Fargus-saf_2.12%2F3.1.2%2Fargus-saf_2.12-3.1.2-assembly.jar)
3. Download the `DirectLeak1` app from DroidBench 3.0: https://github.com/secure-software-engineering/DroidBench/raw/develop/apk/AndroidSpecific/DirectLeak1.apk
4. Setup a configuration
   - Create file `config_amandroid.xml` located in the directory of the AQL-System
   - Copy and Paste the following content:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<config>
<androidPlatforms>/path/to/android/platforms/</androidPlatforms>
<maxMemory>8</maxMemory>
<tools>
   <tool name="Amandroid" version="1">
       <priority>1</priority>
       <execute>
           <run>/path/to/Amandroid/aqlRun.sh %APP_APK% %MEMORY%</run>
           <result>/path/to/Amandroid/outputPath/%APP_APK_FILENAME%/result/AppData.txt</result>
           <instances>0</instances>
           <memoryPerInstance>4</memoryPerInstance>
       </execute>
       <path>/path/to/Amandroid</path>
       <questions>IntraAppFlows</questions>
       <runOnExit>/path/to/BREW/flushMemory.sh</runOnExit>
       <runOnAbort>/path/to/BREW/killpid.sh %PID%</runOnAbort>
   </tool>
</tools>
</config>
```

   - Adjust the path to the Android SDK's platforms directory (`<androidPlatforms>/path/to/android/platforms/</androidPlatforms>`)
   - Adjust the path for Amandroid (`<path>/path/to/Amandroid</path>`) (The directory should contain the previously downloaded .jar file.)
   - Use the same path in `<run>` and `<result>`
   - Adjust the path to flushMemory.sh and killpid.sh to the path of the AQL-System in `<runOnExit>` and `<runOnAbort>`.
   - Lastly adjust `<maxMemory>` and `<memoryPerInstance>`. The latter has to be less than or equal to the first value. Both values are given in gigabytes. (If sufficient memory is provided, a tool might be executed multiple times in parallel.)
5. Make `flushMemory.sh` and `killpid.sh`, located in the AQL-Systems directory, executeable:

```
chmod u+x flushMemory.sh killpid.sh
```

6. Create launch script

```
cd /path/to/Amandroid
nano aqlRun.sh
```

7. Copy and Paste the following:

```
#!/bin/bash
rm -R outputPath
java -Xmx${2}g -jar argus-saf_2.12-3.1.2-assembly.jar t -o outputPath ${1}
```

8. Save *(Ctrl+o)* and exit *(Ctrl+x)* nano

9. Make the script executable:

   ```
   chmod u+x aqlRun.sh
   ```

10. Finally, launch the AQL-System:

    ```
    cd /path/to/AQL-System
    java -jar AQL-System-1.1.1.jar -config config_amandroid.xml -d detailed -gui
    ```

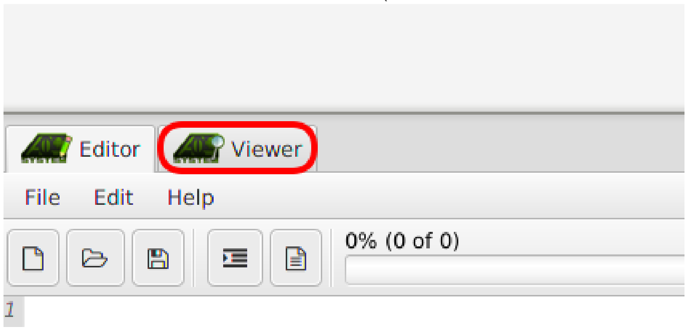
11. Type in a query:

    ```
    Flows IN App('/path/to/DirectLeak1.apk') ?
    ```

12. Click on *Ask query (Green play button on the right in the toolbar)*.

13. Wait for Amandroid to finish its execution.

14. View the AQL-Answer in "Viewer" tab (More information about AQL-Answers can be found: here)



Install & Compile

Install

To simply install the AQL-System you must

- download the current release: here
- and unzip it
- done!

For a *hello world* like tutorial follow the runthrough tutorial.

Compile

To compile the AQL-System by yourself follow these steps:

- Clone the repository
- Build the Maven project by:

  ```
  cd /path/to/project/AQL-System
  mvn
  ```

  (Test might not be completely up-to-date, consider skipping: `mvn -DskipTests`)

Develop

Include the AQL-System as a library.
Therefore download the current release: here
Or add it as Maven dependency:

```
<dependency>
    <groupId>de.foellix</groupId>
    <artifactId>AQL-System</artifactId>
    <version>1.2.0</version>
</dependency>
```

Using the AQL

The following code shows a very simple example. One AQL-Answer is parsed, edited and returned as an Java object.

```
import java.io.File;

import de.foellix.aql.datastructure.Answer;
import de.foellix.aql.datastructure.Flow;
import de.foellix.aql.datastructure.handler.AnswerHandler;

public class AQLUser {
    public Answer use(File answerFile) {
        // Parse an AQL-Answer
        Answer answer = AnswerHandler.parseXML(answerFile);

        // Create a new flow
        Flow flow = new Flow();
        ...

        // Add it to the answer
        answer.getFlows().getFlow().add(flow);

        return answer;
    }
}
```

Using the AQL-System

Another simple example. An AQL-System is initialized. A query is issued and the AQL-Answer produced upon is returned.

```java
import java.io.File;
import java.util.Collection;

import de.foellix.aql.datastructure.Answer;
import de.foellix.aql.system.System;

public class AQLUser {
    public Collection<Answer> query(File appFile) {
        // Initialize aqlSystem
        System aqlSystem = new System();

        // Execute a query
        return aqlSystem.queryAndWait("Flows IN App('" + appFile.getAbsolutePath() + "')");
    }
}
```

Important Classes

The following table hightlights some important classes that may become useful while developing with the AQL-System.

Helpers & Handlers

Class	Capabilities
AnswerHandler	Parsing and dealing with AQL-Answers
ConfigHandler	Parsing and using configurations
Helper	Contains, for example, all toString() methods for generated classes such as AQL-Answers
HashHelper	For generating hashes
EqualsHelper, EqualsOptions	For equals operations on generated classes such as AQL-Answers

Hooks

Along with version 1.2.0 hooks are introduced. You can execute your own code before and after a task (e.g. analysis tool or preprocessor execution) is completed:

```java
import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import de.foellix.aql.Log;
import de.foellix.aql.config.ConfigHandler;
import de.foellix.aql.config.Tool;
import de.foellix.aql.datastructure.Answer;
import de.foellix.aql.system.System;
import de.foellix.aql.system.task.ITaskHook;
import de.foellix.aql.system.task.TaskInfo;

public class HookExample {
    public Collection<Answer> query(File appFile) {
        // Initialize aqlSystem
        System aqlSystem = new System();

        // Apply hook
        Tool awesomeDroid = ConfigHandler.getInstance().getToolByName("AwesomeDroid");
        ITaskHook hook = new AwesomeHook();
        List<ITaskHook> hooks = new ArrayList<>();
        hooks.add(hook);
        aqlSystem.getTaskHooksBefore().getHooks().put(awesomeDroid, hooks);

        // Execute a query
        return aqlSystem.queryAndWait("Flows IN App('" + appFile.getAbsolutePath() + "')");
    }

    private class AwesomeHook implements ITaskHook {
        @Override
        public void execute(TaskInfo taskinfo) {
            Log.msg("Awesome!", Log.DEBUG);
        }
    }
}
```

Launch parameters

# Launch Parameters The AQL-System can be launched with the parameters mentioned in the table below.

Parameter	Meaning
`-help`, `-h`, `-?`, `-man`, `-manpage`	Outputs a very brief manual, which contains a list of all available parameters.
`-query "X"`, `-q "X"`	This parameter is used to assign an AQL-Query. `X` refers to this query.
`-config "X"`, `-cfg "X"`, `-c "X"`	By default the `config.xml` file in the tool's directory is used as configuration. With this parameter a different configuration file can be chosen. `X` has to reference the path to and the configuration file itself.
`-output "X"`, `-out "X"`, `-o "X"`	The answer to a query is automatically saved in the `answers` directory. This parameter can be used to store it in a second file. `X` has to define this file by path and filename.
`-preferexecute`, `-pe`	This parameter is used to force the execution of analysis tools even if a similar question has been asked before.
`-timeout "X"s/m/h`, `-t "X"s/m/h`	With this parameter the maximum execution time of each tool can be set. If it expires a tool's execution is aborted. `X` refers to this time in seconds (e.g. 10s), minutes or hours.
`-debug "X"`, `-d "X"`	The output generated during the execution of this tool can be set to different levels. `X` may be set to: `error`, `warning`, `normal`, `debug`, `detailed` (ascending precision from left to right). Additionally it can be set to `short`, the output will then be equal to `normal` but shorter at some points. By default it is set to `normal`.
`-configwizard`, `-cw`	If this parameter is applied the Config Wizard will be started at the beginning.
`-gui`	If this or no parameters at all are provided the graphical user interface is started. It allows to formulate queries and display answers in a handy way.
`-backup`, `-b`	When this launch parameter is provided, the current storage of the AQL-System is backed up on start.
`-reset`, `-r`	By this parameter the storage of the AQL-System is resetted on start. Whenever a backup is scheduled as well, it will be generated before the reset.

Configuration

Configuration

Any AQL-System has to be configured via an *.xml* file. Its structure is described by this XML Schema Definition file. Such a configuration defines a few environmental properties and most importantly which analysis tools, preprocessors, operators and converters are available in addition to the default operators and converters. Any of these is represented by a `<tool>` element inside the configuration file. There exists two possibilities to create or edit a configuration:

- Edit the .xml file directly
- Use the Configuration Wizard

Option 1: Edit the .xml file directly

The following code shows a basic, shortened version of a configuration:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<config>
    <!-- Environment -->
    <androidPlatforms>path/to/android/platforms</androidPlatforms>
    <maxMemory>6</maxMemory>

    <!-- Tools -->
    <tools>...</tools>
    <preprocessors>...</preprocessors>
    <operators>...</operators>
    <converters>...</converters>
</config>
```

- Inside the `<androidPlatforms>` tag the path to the android platform files has to be specified.
- The memory element tells the AQL-System how much memory shall be used at most.
- `<tools>`, `<preprocessors>`, `<operators>` and `<converters>` each contain a list of `<tool>` elements. Each `<tool>` element describes one analysis tool, preprocessor, operator or converter, respectively. In the following the differences and commonalities of these four are described along with one example for each type.

Analysis tools

```
<tool name="AwesomeDroid" version="1.3.3.7" external="false">
    <priority>0</priority>
    <priority feature="TEST">2</priority>

    <execute>
        <run>/path/to/AwesomeDroid/run.sh %MEMORY% %ANDROID_PLATFORMS% %APP_APK%</run>
        <result>/path/to/AwesomeDroid/results/%APP_APK_FILENAME%_result.txt</result>
        <instances>0</instances>
        <memoryPerInstance>4</memoryPerInstance>
    </execute>

    <path>/path/to/AwesomeDroid</path>
    <questions>IntraAppFlow</questions>

    <runOnExit>/path/to/AQL-System/flushMemory.sh</runOnExit>
    <runOnSuccess>/path/to/AwesomeDroid/success.sh</runOnSuccess>
    <runOnFail>/path/to/AwesomeDroid/fail.sh</runOnFail>
    <runOnAbort>/path/to/AQL-System/killpid.sh %PID%</runOnAbort>
</tool>
```

- Any tool is identified by the values assigned to the two attributes `name` and `version` (In this example the tool's name is `AwesomeDroid` and its version is `1.3.3.7`).
- `<priority>`: If there are two or more tools available which are capable of answering the same AQL-Questions the priority decides which tool is executed.
    - There may be multiple `<priority>` elements, however, only one without a feature attribute. In the example AwesomeDroid has a priority of `0`, which becomes `2` if the associated AQL-Question assigns the feature `TEST`.

How to run any tool is specified through the following elements:

- `<path>`: This describes a path to the directory where the tool shall be executed
- `<run>`: The run tag describes how to call a certain tool by defining the location of a bash script, for example
- `<result>`: This describes where the result can be found once a tool finishes successfully. *(A *-symbol can be used inside this tag to reference an arbitrary substring.)*

Variables, which can be used in the three previously described elements are:

Variable	Meaning
%APP_APK%	The .apk file referenced in an AQL-Question
%APP_APK_FILENAME%	The filename of the .apk file without path and ending
%APP_NAME%	The app's name specified in its manifest
%APP_PACKAGE%	The app's package specified in its manifest
%ANDROID_PLATFORMS%	The Android platforms folder (Specified through)
%MEMORY%	The memory available to an instance of a tool (Specified through)
%PID%	The tools process ID during execution

- `<questions>`: The content of this tag describes which AQL-Questions can be answered with the associated tool. The following options are available (Exemplary associated AQL-Questions can be found in the brackets behind each option):
  - Permissions (`Permissions IN App('A.apk') ?`)
  - Intents (`Intents IN App('A.apk') ?`)
  - IntentFilters (`IntentFilters IN App('A.apk') ?`)
  - IntentSources (`IntentSources IN App('A.apk') ?`)
  - IntentSinks (`IntentSinks IN App('A.apk') ?`)
  - IntraAppFlows (`Flows IN App('A.apk') ?`)
  - InterAppFlows (`Flows FROM App('A.apk') TO App('B.apk') ?`)
- `<instances>`: This element defines how often the associated tool can be executed at the same time.
- `<memoryPerInstance>`: This tag defines how much memory is required and provided to each instance of the associated tool.

There are five more elements which optionally can be specified, namely `<runOnEntry>`, `<runOnExit>`, `<runOnSuccess>`, `<runOnFail>` and `<runOnAbort>`. Each refers to a command or a script which will be executed on certain tool events.

- `<runOnEntry>` is run before a tool is executed.
- `<runOnExit>` is always run after tool execution or abortion.
- `<runOnSuccess>` and `<runOnFail>` get executed depending on whether the tool has finished successfully or not.
- `<runOnAbort>` is run if the tool is aborted.

Variables, which can be used in these five previously described elements in turn are:

Variable	Meaning
%MEMORY%	The memory available to an instance of a tool (Specified through)
%PID%	The tools process ID during execution

Preprocessors

```xml
<tool name="AwesomePreprocessor" version="1.3.3.8" external="false">
    <priority>0</priority>

    <execute>
        <run>/path/to/AwesomePreprocessor/run.sh %APP_APK%</run>
        <result>/path/to/AwesomePreprocessor/results/%APP_APK_FILENAME%_preprocessed.apk</result>
        <instances>0</instances>
        <memoryPerInstance>4</memoryPerInstance>
    </execute>

    <path>/path/to/AwesomePreprocessor</path>
    <questions>TEST</questions>

    <runOnExit>/path/to/AQL-System/flushMemory.sh</runOnExit>
    <runOnAbort>/path/to/AQL-System/killpid.sh %PID%</runOnAbort>
</tool>
```

Preprocessors are specified the same way as analysis tools with one exceptions: The `<questions>` element now holds a list of keywords, separated by `,`, assigned to the associated preprocessor. In the above example only one keyword is assigned (`TEST`).

Operators

```xml
<tool name="AwesomeOperator" version="1.3.3.9" external="false">
    <priority>0</priority>

    <execute>
        <run>/path/to/AwesomeOperator/run.sh %ANSWERS%</run>
        <result>/path/to/AwesomeOperator/results/%ANSWERSHASH%.xml</result>
        <instances>1</instances>
        <memoryPerInstance>4</memoryPerInstance>
    </execute>

    <path>/path/to/AwesomeOperator</path>
    <questions>CONNECT(*)</questions>

    <runOnExit>/path/to/AQL-System/flushMemory.sh</runOnExit>
    <runOnAbort>/path/to/AQL-System/killpid.sh %PID%</runOnAbort>
</tool>
```

Operators are specified the same way as analysis tools with a few exceptions:

- The variables which can be used in `<path>`, `<run>` and `<result>` are:

Variable	Meaning
%ANSWERS%	Input AQL-Answer files separated by " "
%ANSWERSHASH%	SHA-256-hash of the %ANSWERS%-String
%ANDROID_PLATFORMS%	The Android platforms folder (Specified through)
%MEMORY%	The memory available to an instance of a tool (Specified through)
%PID%	The tools process ID during execution

- The `<questions>` element refers to the operators name and specifies its number of parameters
  In the example `CONNECT(*)` tells us that the default `CONNECT` operator gets overwritten by an operator which takes arbitrary many (`*`) AQL-Answers as input.

Converters

```xml
<tool name="AwesomeDroidConverter" version="1.3.3.7" external="false">
    <execute>
        <run>/path/to/AwesomeDroidConverter/run.sh %RESULT_FILE% results/%APP_APK_FILENAME%.xml</run>
        <result>/path/to/AwesomeDroidConverter/results/%APP_APK_FILENAME%.xml</result>
        <instances>0</instances>
        <memoryPerInstance>4</memoryPerInstance>
    </execute>

    <path>/path/to/AwesomeDroidConverter</path>
    <questions>AwesomeDroid</questions>
</tool>
```

Converters again are specified the same way as analysis tools with some exceptions:

- One additional variable can be used in `<path>` and `<run>`:

Variable	Meaning
%RESULT_FILE%	Result file of the associated analysis tool

- The `<questions>` element in this case refers to the analysis tools (separated by `,`) associated with this converter. (In the example only `AwesomeDroid` is associated.)
- The elements `<runOnEntry>`, `<runOnExit>`, `<runOnSuccess>`, `<runOnFail>` and `<runOnAbort>` are not supported for converters, yet.

Complete Example

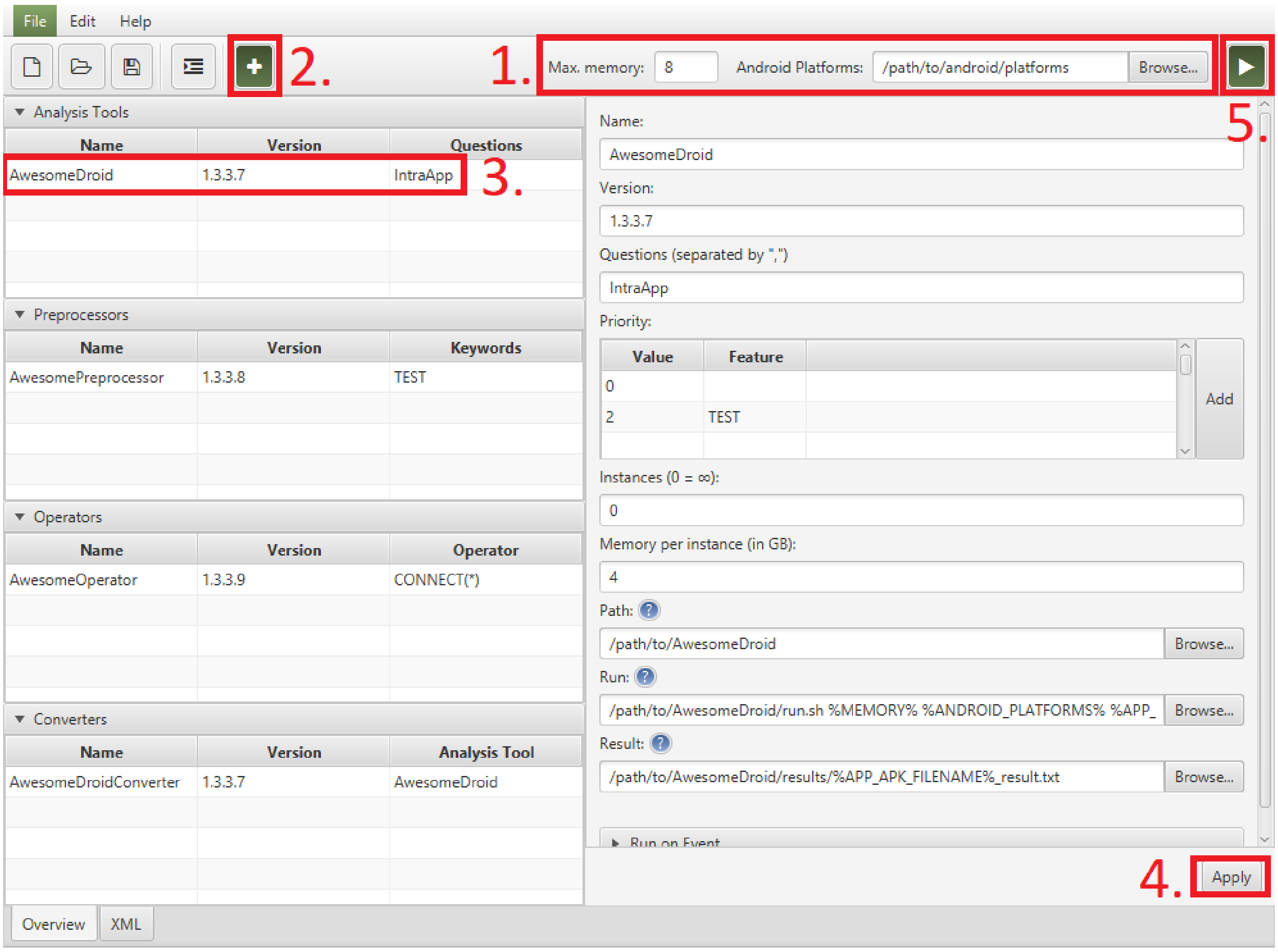
```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<config>
    <androidPlatforms>path/to/android/platforms</androidPlatforms>
    <maxMemory>8</maxMemory>
    <tools>
        <tool name="AwesomeDroid" version="1.3.3.7" external="false">
            <priority>0</priority>
            <priority feature="TEST">2</priority>
            <path>/path/to/AwesomeDroid</path>
            <execute>
                <run>/path/to/AwesomeDroid/run.sh %MEMORY% %ANDROID_PLATFORMS% %APP_APK%</run>
                <result>/path/to/AwesomeDroid/results/%APP_APK_FILENAME%_result.txt</result>
                <instances>0</instances>
                <memoryPerInstance>4</memoryPerInstance>
            </execute>
            <runOnExit>/path/to/AQL-System/flushMemory.sh</runOnExit>
            <runOnSuccess>/path/to/AwesomeDroid/success.sh</runOnSuccess>
            <runOnFail>/path/to/AwesomeDroid/fail.sh</runOnFail>
            <runOnAbort>/path/to/AQL-System/killpid.sh %PID%</runOnAbort>
            <questions>IntraApp</questions>
        </tool>
    </tools>
    <preprocessors>
        <tool name="AwesomePreprocessor" version="1.3.3.8" external="false">
            <priority>0</priority>
            <path>/path/to/AwesomePreprocessor</path>
            <execute>
                <run>/path/to/AwesomePreprocessor/run.sh %APP_APK%</run>
                <result>/path/to/AwesomePreprocessor/results/%APP_APK_FILENAME%_preprocessed.apk</result>
                <instances>0</instances>
                <memoryPerInstance>4</memoryPerInstance>
            </execute>
            <runOnExit>/path/to/AQL-System/flushMemory.sh</runOnExit>
            <runOnAbort>/path/to/AQL-System/killpid.sh %PID%</runOnAbort>
            <questions>TEST</questions>
        </tool>
    </preprocessors>
    <operators>
        <tool name="AwesomeOperator" version="1.3.3.9" external="false">
            <priority>0</priority>
            <path>/path/to/AwesomeOperator</path>
            <execute>
                <run>/path/to/AwesomeOperator/run.sh %ANSWERS%</run>
                <result>/path/to/AwesomeOperator/results/%ANSWERSHASH%.xml</result>
                <instances>1</instances>
                <memoryPerInstance>4</memoryPerInstance>
            </execute>
            <runOnExit>/path/to/AQL-System/flushMemory.sh</runOnExit>
            <runOnAbort>/path/to/AQL-System/killpid.sh %PID%</runOnAbort>
            <questions>CONNECT(*)</questions>
        </tool>
    </operators>
    <converters>
        <tool name="AwesomeDroidConverter" version="1.3.3.7" external="false">
            <path>/path/to/AwesomeDroidConverter</path>
            <execute>
                <run>/path/to/AwesomeDroidConverter/run.sh %RESULT_FILE% results/%APP_APK_FILENAME%.xml</run>
                <result>/path/to/AwesomeDroidConverter/results/%APP_APK_FILENAME%.xml</result>
                <instances>0</instances>
                <memoryPerInstance>4</memoryPerInstance>
            </execute>
            <questions>AwesomeDroid</questions>
        </tool>
    </converters>
</config>
```

Option 2: Use the Config Wizard

You find the *ConfigWizard* in the *Help* menu of the GUI.
Alternatively you can launch it as follows:

```
java -jar AQL-System-1.2.0 -cw
```

The screenshot below shows the Config Wizard. All elements explained before can easily be edited here as well.



Max. memory: 8 Android Platforms: /path/to/android/platforms

▼ Analysis Tools

Name	Version	Questions
AwesomeDroid	1.3.3.7	IntraApp

▼ Preprocessors

Name	Version	Keywords
AwesomePreprocessor	1.3.3.8	TEST

▼ Operators

Name	Version	Operator
AwesomeOperator	1.3.3.9	CONNECT(*)

▼ Converters

Name	Version	Analysis Tool
AwesomeDroidConverter	1.3.3.7	AwesomeDroid

Name: AwesomeDroid
Version: 1.3.3.7
Questions (separated by ","): IntraApp
Priority:

Value	Feature
0	
2	TEST

Instances (0 = ∞): 0
Memory per instance (in GB): 4
Path: /path/to/AwesomeDroid
Run: /path/to/AwesomeDroid/run.sh %MEMORY% %ANDROID_PLATFORMS% %APP_
Result: /path/to/AwesomeDroid/results/%APP_APK_FILENAME%_result.txt
▸ Run on Event

- The environmental properties can be defined at **1.**
- New tools of any kind can be added at **2.**
- To edit a tool:
  - Select it, for example by clicking on **3.**
  - Edit its properties on the right hand side
  - Apply the changed by clicking at **4.**
- To continue with the configuration you have set up click on **5.**

Query execution

Query Execution

An AQL-Query can be executed using the AQL-System via

- a command line interface
  OR
- a graphical user interface

Anyway it has to be configured first (see the configuration tutorial).

Command Line Interface (CLI)

To execute a query via command line, launch the AQL-System `java -jar AQL-System-1.2.0.jar` with the `-query` parameter and provide a query. The following example issues a query that asks for Flows inside one app (`A.apk`):

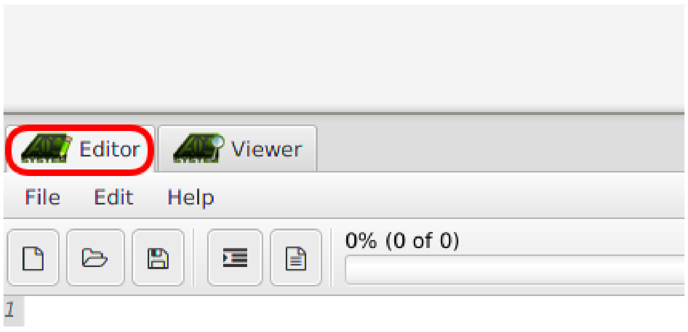
```
java -jar /path/to/AQL-System-1.2.0.jar -query ”Flows IN App('A.apk') ?”
```

Graphical User Interface (GUI)

- To execute a query via the GUI, launch it first:

  ```
  java -jar /path/to/AQL-System-1.2.0.jar -gui
  ```

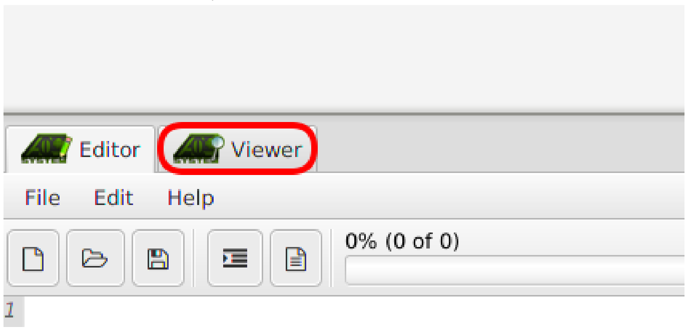
- Then switch to the editor tab:



- Type in an AQL-Query inside the editor window:

  ```
  Flows IN App('A.apk') ?
  ```

- Click "Run" *(Green play button at the right in the toolbar)*.
- Wait for the AQL-System to compute the answer
- To view AQL answer, switch to viewer tab



- AQL-Answers are provided in two views:
  - textual view: Allows to view and edit the .xml file representing the AQL-Answer.

- graphical view: Shows the flows, permissions, intent-sources and -sinks in a single graph.

Add your tools

Add tools

In order to add a new analysis tool, preprocessor or operator to the AQL-System, its configuration has to be adapted (See the configuration tutorial). Adding an analysis tool may require a new converter to be added as well. To do so, three options are available:

1. Existing Converter

If one of the converters implemented in the AQL-System fits your needs. You just have to assign the associated toolname for the analysis tool in the configuration. Currently the following converters are available:

- Amandroid
- DIALDroid *(If you plan to use this converter, please adapt `data/converter/dialdroid_config.properties`)*
- DidFail
- DroidSafe
- FlowDroid
- FlowDroid2 (> version 2.5.0)
- IC3
- IccTA
- PAndA^2

2. Add new converter (internal)

Implement your own converter and compile the AQL-System on your own (See the install & compile tutorial).

1. Include your converter by implementing the interface `IConverter`
2. Add it to the `ConverterRegistry` inside the `de.foellix.aql.converter` package by adding the following line *(after Line 34 in ConverterRegistry.java)*:

   ```
   this.map.put("AwesomeDroid".toLowerCase(), new AwesomeDroidConverter());
   ```

   (In this example `AwesomeDroid` refers to the toolname of the tool you want to add, which is defined in the configuration. The associated converter is `AwesomeDroidConverter`.)

3. Add new converter (external)

1. Develop your own converter...
   - taking the result file of the tool you want to add as input
   - and generating an AQL-Answer as output.
2. Specify this converter in the configuration (See the configuration tutorial).
   - Make sure to assign the toolname of all tools, for which the converter should be used, in `<questions>`.
     (e.g. `<questions>AwesomeDroid1, AwesomeDroid2</questions>`)

AQL-System 1.2.0

The Android App Analysis Query Language - System (AQL-System)

The AQL consists of two main parts, namely AQL-Queries (compositions of AQL-Questions) and AQL-Answers. The AQL-System takes AQL-Queries as input and outputs AQL-Answers.

New Tutorials

Along with the release of version 1.2.0 configurations must be upgraded:
Configuration Upgrades

Basic Tutorials

Runthrough
Install & Compile & Develop
Launch parameters
Configuration
Query execution
Add your tools

FAQ

Configuration Upgrades

Configuration Upgrades

Along with the release of version 1.2.0 the structure of configurations has changed (see XML Schema Definition file).

The AQL-System now allows to configure external tools:

```xml
<tool name="AwesomeDroidExternal" version="1.3.3.7" external="true">
    <priority>1</priority>
    <execute>
        <url>http://awesome.droid.com:1337/AQL-WebService/ask</url>
        <username>droid</username>
        <password>****</password>
    </execute>
    <path>/path/to/directory/for/temporary/files</path>
    <questions>IntraAppFlows</questions>
</tool>
```

For internal tools the following is required:

```xml
<tool name="AwesomeDroidInternal" version="1.3.3.7" external="false">
    <priority>1</priority>
    <execute>
        <run>/path/to/AwesomeDroid/run.sh %MEMORY% %ANDROID_PLATFORMS% %APP_APK%</run>
        <result>/path/to/AwesomeDroid/results/%APP_APK_FILENAME%_result.txt</result>
        <instances>0</instances>
        <memoryPerInstance>4</memoryPerInstance>
    </execute>
    <path>/path/to/AwesomeDroid</path>
    <questions>IntraAppFlow</questions>
</tool>
```

Thus, the attribute `external="false"` has to be set and the element `<execute>` must be specified.

Remark: Old configurations had no `<execute>` element. However, all subelements for internal tools should already be existent.

FAQ

FAQ

No questions, yet!