

---

# **smalisca Documentation**

***Release 0.2***

**Victor Dorneanu**

June 21, 2015



<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>Screenshots</b>	<b>5</b>
<b>3</b>	<b>Installation</b>	<b>7</b>
<b>4</b>	<b>How to use it</b>	<b>9</b>
4.1	First touch . . . . .	9
4.2	Parsing . . . . .	10
4.3	Analyzing . . . . .	10
4.4	Drawing . . . . .	11
4.5	Contents . . . . .	11
<b>5</b>	<b>Indices and tables</b>	<b>41</b>



If you ever have looked at Android applications you know to appreciate the ability of analyzing your target at the most advanced level. Dynamic program analysis will give you a pretty good overview of your applications activities and general behaviour. However sometimes you'll want to just analyze your application **without** running it. You'll want to have a look at its components, analyze how they interact and how data is tainted from one point to another.

This is was the major factor driving the development of *smalisca*. There are indeed some good reasons for a *static code analysis* before the *dynamic* one. Before interacting with the application I like to know how the application has been build, if there is any API and generate all sort of *call flow graphs*. In fact graphs have been very important to me since they *visualize* things. Instead of jumping from file to file, from class to class, I just look at the graphs.

While graph building has been an important reason for me to code such a tool, *smalisca* has some other neat **features** you should read about.



---

## Features

---

At the moment there are some few major functionalities like:

- **parsing**

You can parse a whole directory of **Smali** files and **extract**:

- class information
- class properties
- class methods
- calls between methods of different classes

You can then **export** the results as **JSON** or **SQLite**.

Have a look at the [parsing page](#) for more information.

- **analyzing**

After exporting the results you'll get an **interactive prompt** to take a closer look at your parsed data. You can **search** for classes, properties, methods and even method calls. You can then apply several **filters** to your search criterias like:

```
smalisca> sc -c class_name -p test -r 10 -x path -s class_type
```

This command will search for *10* (-r 10) classes which contain the pattern *test* (-p) in their *class name* (-c). Afterwards the command will exclude the column *path* (-x path) from the results and sort them by the *class type* (-s).

Let's have a look at another example:

```
smalisca> scl -fc com/android -fm init -r 10
```

This will search for all **method calls** whose *calling* class name contains the pattern *com/android* (-fc). Additionally we can look for calls originating from methods whose name contain the pattern *init* (-fm).

You can of course read your commands from a file and analyze your results in a *batch*- like manner:

```
$ cat cmd.txt
sc -c class_name -p com/gmail/xlibs -r 10 -x path
quit
$ ./smalisca.py analyzer -i results.sqlite -f sqlite -c cmd.txt
...
```

*Addition in version 0.2:* You can access the results via a web API.

Have a look at the [analysis page](#) for more information.

- **web API**

smalisca provides a REST web service in order to easily interact with the results by just using a web client. This way you can access data in your own (fancy) web application and have a clean separation between backend and frontend.

Read more about the available REST API at the [web API page](#).

- **visualizing**

I think this the **most** valuable feature of *smalisca*. The ability to visualize your results in a structured way makes your life more comfortable. Depending on what you're interested in, this tool has several graph drawing features I'd like to promote.

At first you can draw your packages including their classes, properties and methods:

```
smalisca> dc -c class_name -p test -f dot -o /tmp/classes.dot
:: INFO      Wrote results to /tmp/classes.dot
smalisca>
```

This will first search classes whose class name contains *test* and then export the results in the **Graphviz DOT** language. You can then manually generate a graph using *dot*, *neato*, *circo* etc. Or you do that using the interactive prompt:

```
smalisca> dc -c class_name -p test -f pdf -o /tmp/classes.pdf --prog neato
:: INFO      Wrote results to /tmp/classes.pdf
smalisca>
```

Have a look at the [drawing page](#) for more information.



---

## Screenshots

---

Have a look at the *screenshots page*.



---

## Installation

---

Refer to the *installation page*.



## How to use it

After installing the tool, you may want to first pick up an Android application (APK) to play with. Use [apktool](#) or my own tool [ADUS](#) to dump the APKs content. For the sake of simplicity I'll be using **FakeBanker** which I've analyzed in a previous [blog post](#).

## 4.1 First touch

But first let's have a look at the tools main options:

```
$ smalisca --help
```

The diagram is a complex geometric structure composed of multiple rows of connected line segments. It features a series of peaks and valleys, creating a jagged, mountain-like profile. The segments are primarily horizontal and diagonal, with some segments labeled with letters such as 'L' and 'U'. The overall structure is symmetrical and intricate, with many small, repeating geometric units.

```
:: Author:      Victor <Cyneox> Dorneanu
:: Desc:       Static Code Analysis tool for Smali files
:: URL:        http://nullsecurity.net, http://{blog,www}.dorneanu
:: Version:    0.2
```

```
usage: smalisca (sub-commands ...) [options ...] {arguments ...}
```

```
[--] Static Code Analysis (SCA) tool for Baskmali (Smali) files.
```

commands:

analyzer

```
[--] Analyze results using an interactive prompt or on the command line.
```

parser

```
[--] Parse files and extract data based on Smali syntax.
```

web

```
[--] Analyze results using web API.
```

optional arguments:

```
-h, --help          show this help message and exit
--debug            toggle debug output
--quiet           suppress all output
--log-level {debug,info,warn,error,critical}
                  Change logging level (Default: info)
-v, --version      show program's version number and exit
```

## 4.2 Parsing

I'll first **parse** some directory for **Smali** files before doing the analysis stuff:

```
$ smalisca parser -l ~/tmp/FakeBanker2/dumped/smali -s java -f sqlite -o fakebanker.sqlite

...

:: INFO      Parsing .java files in /home/victor/tmp/FakeBanker2/dumped/smali ...
:: INFO      Finished parsing!
:: INFO      Exporting results to SQLite
:: INFO      Extract classes ...
:: INFO      Extract class properties ...
:: INFO      Extract class methods ...
:: INFO      Extract calls ...
:: INFO      Commit changes to SQLite DB
:: INFO      Wrote results to fakebanker.sqlite
:: INFO      Finished scanning
```

Also have a look at the [parsing page](#) for further information.

## 4.3 Analyzing

Now you're free to do whatever you want with your generated exports. You can inspect the **SQLite DB** directly or use *smaliscas* **analysis** features:

```
$ smalisca analyzer -f sqlite -i fakebanker.sqlite

...
```

```
smalisca>sc -x path -r 10
```

id	class_name	class
1	Landroid/support/v4/net/ConnectivityManagerCompat	public
2	Landroid/support/v4/view/AccessibilityDelegateCompat\$AccessibilityDelegateJellyBeanImpl	
3	Landroid/support/v4/view/ViewCompat\$ViewCompatImpl	internal
4	Landroid/support/v4/app/ActivityCompatHoneycomb	
5	Landroid/support/v4/app/NoSaveStateFrameLayout	
6	Landroid/support/v4/net/ConnectivityManagerCompatHoneycombMR2	
7	Lcom/gmail/xdroid/BuildConfig	public
8	Landroid/support/v4/app/BackStackRecord\$Op	final
9	Landroid/support/v4/app/FragmentManagerImpl	final
10	Landroid/support/v4/app/ShareCompat\$ShareCompatImpl	internal

Also refer to the *analysis page* for more available **commands** and options.

## 4.4 Drawing

Please refer to the *drawing page* for full examples.

## 4.5 Contents

### 4.5.1 Installation

The installation process is quite straight forward. Download tool from [GitHub](#) or clone the whole project locally:

```
$ git clone https://github.com/dorneanu/smalisca
...
```

#### Virtual environment

You may now want to setup a virtual local environment:

```
$ mkdir env
$ virtualenv env
...
$ source env/bin/activate
```

#### Install package

Now into the packages root directory and install the package:

```
$ cd smalisca $ make install ...
```

#### Using PyPI

smalisca is also available at PyPI. You may want to install it using:

```
$ pip install smalisca
```

That's it! Now you're ready to run *smalisca*.

#### Uninstall package

From the root directory run:

```
$ cd smalisca
$ make uninstall
```

### 4.5.2 Parsing files

As a very first step before conducting any analysis, you'll have to parse your Smali files for valuable information like:

- class names
- class properties
- class methods
- method calls

Once you have extracted this information from the files, you're ready to go with the analysis. Using the **parser** sub-command you can parse a directory for Smali files:

```
$ smalisca parser --help
...

usage: smalisca (sub-commands ...) [options ...] {arguments ...}

[--] Parse files and extract data based on Smali syntax.

optional arguments:
  -h, --help                show this help message and exit
  --debug                  toggle debug output
  --quiet                  suppress all output
  --log-level {debug,info,warn,error,critical}
                          Change logging level (Default: info)
  -j JOBS, --jobs JOBS    Number of jobs/processes to be used
  -l LOCATION, --location LOCATION
                          Set location (required)
  -d DEPTH, --depth DEPTH
                          Path location depth
  -s SUFFIX, --suffix SUFFIX
                          Set file suffix (required)
  -f {json,sqlite}, --format {json,sqlite}
                          Files format
  -o OUTPUT, --output OUTPUT
                          Specify output file
```

The most important options for parsing are the *location* (-l) and the *suffix* (-s). For exporting the results you'll have to specify the *output format* (-f) and the corresponding *output file* (-o).

---

**Note:** Make sure you **delete** the sqlite file before re-running the parser. Otherwise you might get confronted with DB errors.

---

Example:

```
$ smalisca -l /tmp/APK-dumped -s java -f sqlite -o apk.sqlite
...
```

### Concurrency

In order to improve performance concurrency has been added to smalisca since *version 0.2*. You can read more in this [blog post](#) about the problems I have encountered and their solution.

Using the *-d* (depth) parameter you can now specify up to which **folder depth** smalisca should look up for files and directories. When a file list has been identified for parsing then you can use the *-j* (jobs) parameter to split your initial



list into multiple sub-lists. Afterwards for every sub-list a new process will be spawned and the parsing stuff can then take place. Just say we have following folder structure:

```
a
-- b1
|   -- c1
|   -- c2
-- b2
|   -- d1
|   -- d2
-- b3
    -- e1
    -- e2
    -- e3
```

You can now either:

1. Parse files in directories: c1, c2, d1, d2, e1, e2, e3 + sub-folders
2. Parse files in directories: b1, b2, b3 + sub-folders
3. Parse in folder a + sub-folders

Depending on the scenario you would then specify like this:

1. `smalisca ... -d 3...`
2. `smalisca ... -d 2 ...`
3. `smalisca ... -d 1 ...`

---

**Note:** Concurrency is available in smalisca since version 0.2.

---

## Create new parser

You may also want to parse the files programmatically

```
from smalisca.core.smalisca_main import SmaliscaApp
from smalisca.modules.module_smali_parser import SmaliParser

# Create new smalisca app
# You'll have to create a new app in order to set logging
# settings and so on.
app = SmaliscaApp()
app.setup()

# Set log level
app.log.set_level('info')

# Specify the location where your APK has been dumped
location = '/tmp/APK-dumped'

# Specify file name suffix
suffix = 'java'

# Create a new parser
parser = SmaliParser(location, suffix)

# Go for it!
parser.run()
```

```
# Get results
results = parser.get_results()
```

### 4.5.3 Analyze results

#### Basic usage

After having parsed and collected valuable information about your application, you're ready to go with the analysis stuff.

---

**Note:** At the moment only SQLite analysis is supported.

---

Your previously generated results should be located in a *SQLite* DB. But first let's have a look at the main options:

```
$ smalisca analyzer --help

...

usage: smalisca (sub-commands ...) [options ...] {arguments ...}

[--] Analyze results using an interactive prompt or on the command line.

optional arguments:
  -h, --help            show this help message and exit
  --debug              toggle debug output
  --quiet              suppress all output
  --log-level {debug,info,warn,error,critical}
                        Change logging level (Default: info)
  --config CONFIG_FILE Specify config file
  -i FILENAME, --input FILENAME
                        Specify results file to read from
  -f {sqlite}, --format {sqlite}
                        Files format
  -c COMMANDS_FILE      Read commands from file instead of interactive prompt
```

At the moment there are 2 ways how to interact with the results:

- using the provided **interactive prompt** (default)
- specifying a **file** containing **commands** to be executed (-c)

---

**Note:** SQL ninjas could of course analyze the SQLite DB directly without any 3rd party applications.

---

#### Interactive

In the **interactive** modus all you have to do is to specify the location of the SQLite DB and the format (which will be of course *sqlite*):

```
$ smalisca analyzer -i /tmp/fakebanker.sqlite -f sqlite

...

:: INFO      Successfully opened SQLite DB
:: INFO      Creating analyzer framework ...
```

```
:: INFO      Starting new analysis shell
```

```
-- Analyzer -----
Welcome to smalisca analyzer shell.
Type ? or help to list available commands.
Type "<command> --help" for additional help.
```

```
smalisca>
```

Now you're ready to interact with the results. Just type **help** to see a list of available *commands*:

```
smalisca>help
```

```
Documented commands (type help <topic>):
-----
dc dcl dxcl help q quit sc scl sm sp sxcl
```

```
smalisca>
```

For every provided type **<command> --help** to see a list of available options:

```
smalisca>sc --help
usage: sc [-h] [-c SEARCH_TYPE] [-p SEARCH_PATTERN] [-s SORTBY] [--reverse]
         [-r RANGE] [--max-width MAX_WIDTH] [-x EXCLUDE_FIELDS]

[--] Search for classes

Specify by '-c' in which column you'd like to search for a pattern (specified by '-p').
Examples:
```

```
a) List available columns
   sc -c ?
```

```
b) Search for pattern "test" in column "class_name" (first 10 results)
   sc -c class_name -p test -r 10
```

```
c) Search for pattern "test2" in column "class_type" (print only from index 10 to 20)
   sc -c class_type -p test2 -r 10,20
```

You can also exclude table fields using '-x':

```
a) Exclude only one column
   sc -c class_type -p test2 -x depth
```

```
b) Exclude multiple columns:
   sc -c class_type -p test2 -x depth,id,class_name
```

optional arguments:

```
-h, --help      show this help message and exit
-c SEARCH_TYPE  Specify column.
                  Type ? for list
-p SEARCH_PATTERN Specify search pattern
-s SORTBY       Sort by column name
--reverse       Reverse sort order
-r RANGE        Specify output range by single integer or separated by ','
--max-width MAX_WIDTH
                  Global column max width
-x EXCLUDE_FIELDS Exclude table fields
smalisca>
```

In this specific case you could run:

```
smalisca>sc -c ?
['id', 'class_name', 'class_type', 'class_package', 'depth', 'path']
No results! :(
smalisca>sc -c class_name -p gmail -x path -r 10
```

id	class_name	class_type	class_package	depth
7	Lcom/gmail/xservices/XService\$MyRun		Lcom.gmail.xservices	4
13	Lcom/gmail/xdpack/R\$id	public final	Lcom.gmail.xpack	4
24	Lcom/gmail/xlibs/myFunctions	public	Lcom.gmail.xlibs	4
35	Lcom/gmail/xdpack/R\$menu	public final	Lcom.gmail.xpack	4
59	Lcom/gmail/xservices/XSmsIncom\$1RequestTask		Lcom.gmail.xservices	4
68	Lcom/gmail/xdpack/R\$raw	public final	Lcom.gmail.xpack	4
69	Lcom/gmail/xlibs/myFunctions\$1RequestTask		Lcom.gmail.xlibs	4
81	Lcom/gmail/xservices/XRepeat\$1RequestTask		Lcom.gmail.xservices	4
88	Lcom/gmail/xdpack/R\$style	public final	Lcom.gmail.xpack	4
97	Lcom/gmail/xbroadcast/OnBootReceiver	public	Lcom.gmail.xbroadcast	4

### Batch like

In the **batch** modules one could provide the commands in a file. These will be executed in that specific order:

```
$ cat cmd.txt
sc -c class_name -p gmail -x path -r 10
quit
$ smalisca analyzer -i /tmp/fakebanker.sqlite -f sqlite -c cmd.txt
```

...

```
:: INFO      Successfully opened SQLite DB
:: INFO      Creating analyzer framework ...
:: INFO      Reading commands from cmd.txt
```

```
-- Analyzer -----
Welcome to smalisca analyzer shell.
Type ? or help to list available commands.
Type "<command> --help" for additional help.
```

id	class_name	class_type	class_package	depth
7	Lcom/gmail/xservices/XService\$MyRun		Lcom.gmail.xservices	4
13	Lcom/gmail/xdpack/R\$id	public final	Lcom.gmail.xpack	4
24	Lcom/gmail/xlibs/myFunctions	public	Lcom.gmail.xlibs	4
35	Lcom/gmail/xdpack/R\$menu	public final	Lcom.gmail.xpack	4
59	Lcom/gmail/xservices/XSmsIncom\$1RequestTask		Lcom.gmail.xservices	4
68	Lcom/gmail/xdpack/R\$raw	public final	Lcom.gmail.xpack	4
69	Lcom/gmail/xlibs/myFunctions\$1RequestTask		Lcom.gmail.xlibs	4
81	Lcom/gmail/xservices/XRepeat\$1RequestTask		Lcom.gmail.xservices	4
88	Lcom/gmail/xdpack/R\$style	public final	Lcom.gmail.xpack	4
97	Lcom/gmail/xbroadcast/OnBootReceiver	public	Lcom.gmail.xbroadcast	4

## Available commands

### s

[S]search for a **pattern** (-p) inside **all** tables. But you can also specify the **table** (-t) you'd like to lookup your pattern:

```
smalisca>s -p decrypt
- Classes -----
:: WARNING      No found classes.

- Properties -----
:: WARNING      No found properties.

- Const strings -----
:: WARNING      No found const strings.

- Methods -----
:: INFO          Found 1 results

:: ID: 131

      [+] Name:      decrypt
      [+] Type:      public
      [+] Args:      Ljava/lang/String;
      [+] Ret:       Ljava/lang/String;
      [+] Class:     Lcom/gmail/xlibs/Blowfish
```

And now specifying the table:

```
smalisca>s -t const -p container=
- Classes -----
:: WARNING      No found classes.

- Properties -----
:: WARNING      No found properties.

- Const strings -----
:: INFO          Found 2 results

:: ID: 39

      [+] Variable:   v0
      [+] Value:      mContainer=
      [+] Class:      Landroid/support/v4/app/Fragment

:: ID: 833

      [+] Variable:   v6
      [+] Value:      mContainer=
      [+] Class:      Landroid/support/v4/app/FragmentManagerImpl
```

```
- Methods -----
:: WARNING      No found methods.
```

**sc**

[S]earch for [c]lasses. You can search for a specific **pattern** (-p) in the available **columns**:

```
smalisca>sc -c ?
['id', 'class_name', 'class_type', 'class_package', 'depth', 'path']
```

Example:

```
smalisca>sc -c class_type -p public
```

**sp**

[S]earch for [p]roperties. You can search for a specific **pattern** (-p) in the available **columns**:

```
smalisca>sp -c ?
['id', 'property_name', 'property_type', 'property_info', 'property_class']
```

Example:

```
smalisca>sp -c property_class -p com/gmail
```

**scs**

[S]earch for [c]onstant [s]trings. You can search for a specific **pattern** (-p) in the available **columns**:

```
smalisca>scs -c ?
['id', 'const_string_var', 'const_string_value', 'const_string_class']
```

Example:

```
smalisca>scs -c const_string_value -p http
+-----+-----+-----+-----+
| id | const_string_var | const_string_value | const_string_class |
+-----+-----+-----+-----+
| 321 | v11              | HTTP              | Lcom/gmail/xlibs/myFunctions |
| 337 | v10              | HTTP              | Lcom/gmail/xlibs/myFunctions |
+-----+-----+-----+-----+
```

**sm**

[S]earch for [m]ethods. You can search for a specific **method** (-m) in the available **columns**:

```
smalisca>sm -c ?
['id', 'method_name', 'method_type', 'method_args', 'method_ret', 'method_class']
```

Example:

```
smlisca>sm -c method_ret -p I
```

**scl**

[S]earch for calls [cl]. Every call has a **source** (class, method) and a **destination** (class, method). Additionally a call can have several **parameters** and a **return** value. Using this command you can apply several **filters** to each call

“component”:

```
smalisca>scl --help
usage: scl [-h] [-fc FROM_CLASS] [-fm FROM_METHOD] [-tc TO_CLASS]
          [-tm TO_METHOD] [-fa LOCAL_ARGS] [-ta DEST_ARGS] [-s SORTBY]
          [--reverse] [-r RANGE] [--max-width MAX_WIDTH] [-x EXCLUDE_FIELDS]
```

>> Search for calls

You can apply filters by using the optional arguments.  
Without any arguments the whole 'calls' table will  
be printed.

optional arguments:

-h, --help	show this help message and exit
-fc FROM_CLASS	Specify calling class (from)
-fm FROM_METHOD	Specify calling method (from)
-tc TO_CLASS	Specify destination class (to)
-tm TO_METHOD	Specify destination method (to)
-fa LOCAL_ARGS	Local arguments (from)
-ta DEST_ARGS	Destination arguments (to)
-s SORTBY	Sort by column name
--reverse	Reverse sort order
-r RANGE	smecify output range by single integer or separated by ','
--max-width MAX_WIDTH	Global column max width
-x EXCLUDE_FIELDS	Exclude table fields

Examples:

```
smalisca>scl -fc com/gmail -fm init -r 10
...
smalisca>scl -tm create
...
```

## sxcl

[S]earch for cross [x] calls [cl]. This command is very similar to the *scl* one and searches for calls as well. *sxcl* allows you to search for calls that:

- *refer* to a class and/or method or
- are *invoked from* a class and/or method

These are the main options:

```
smalisca>sxcl --help
usage: sxcl [-h] [-c CLASS_NAME] [-m METHOD_NAME] -d {to,from}
          [--max-depth [XREF_DEPTH]] [-s SORTBY] [--reverse] [-r RANGE]
          [--max-width MAX_WIDTH] [-x EXCLUDE_FIELDS]
```

>> Search for calls

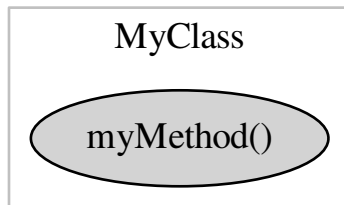
You can apply filters by using the optional arguments.  
Without any arguments the whole 'calls' table will  
be printed.

optional arguments:

-h, --help	show this help message and exit
------------	---------------------------------

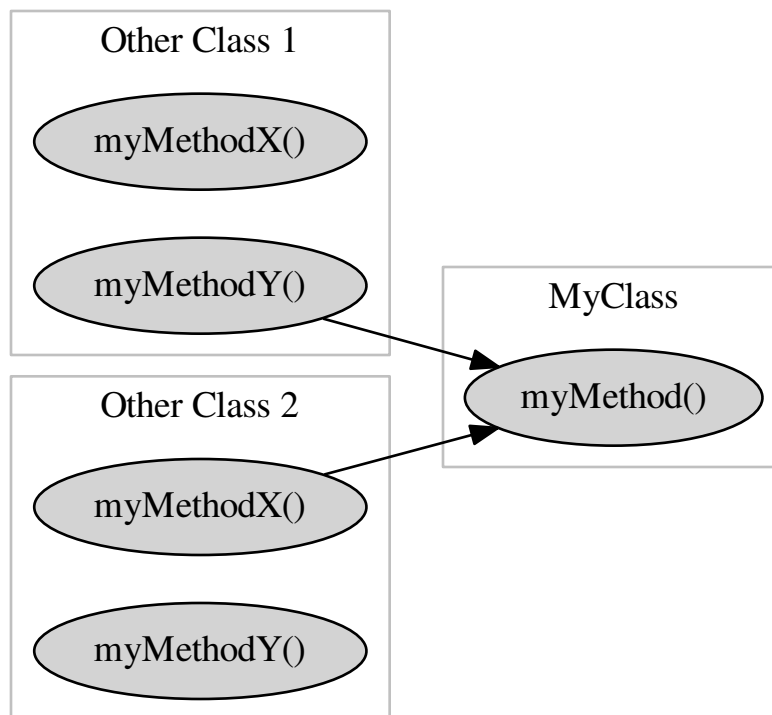
```
-c CLASS_NAME          Specify class name
-m METHOD_NAME          Specify method name
-d {to,from}           Cross-reference direction
--max-depth [XREF_DEPTH]
                        Cross-References max depth
                        Default: 1
-s SORTBY              Sort by column name
--reverse              Reverse sort order
-r RANGE               smecify output range by single integer or separated by ','
--max-width MAX_WIDTH  Global column max width
-x EXCLUDE_FIELDS      Exclude table fields
```

You can specify a *class name* (-c) and/or a *method name* (-m). You can then define the *direction* cross calls should be searched. To give you a better understand what this is about, let's say you have a method *myMethod* in class *MyClass*.

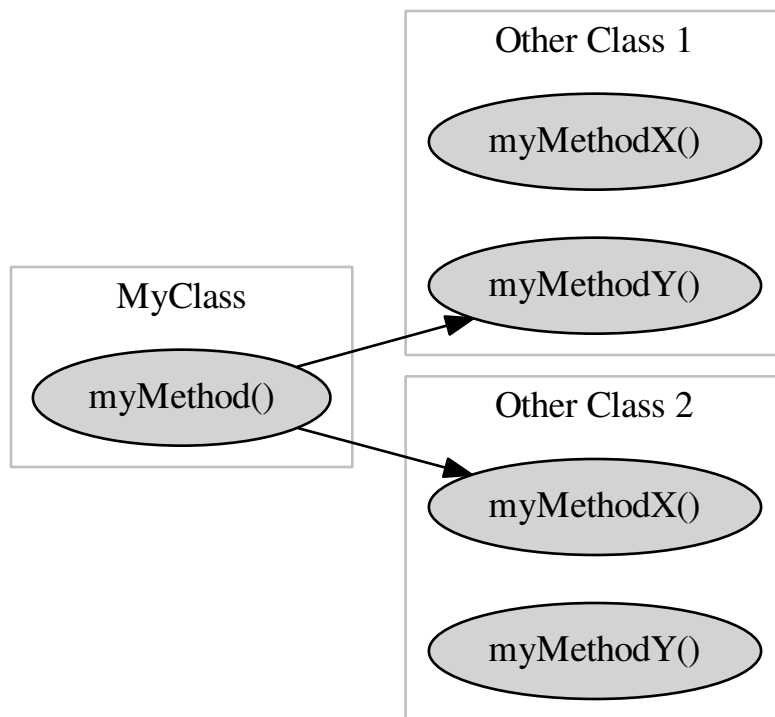


You may now want to find out classes/method which **point to** (-d to) to this class/method:



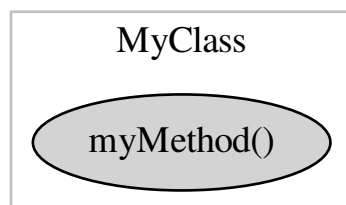


On the other side you may want to search for classes/methods which are **called/invoked** (-d from) by *MyClass/myMethod*:

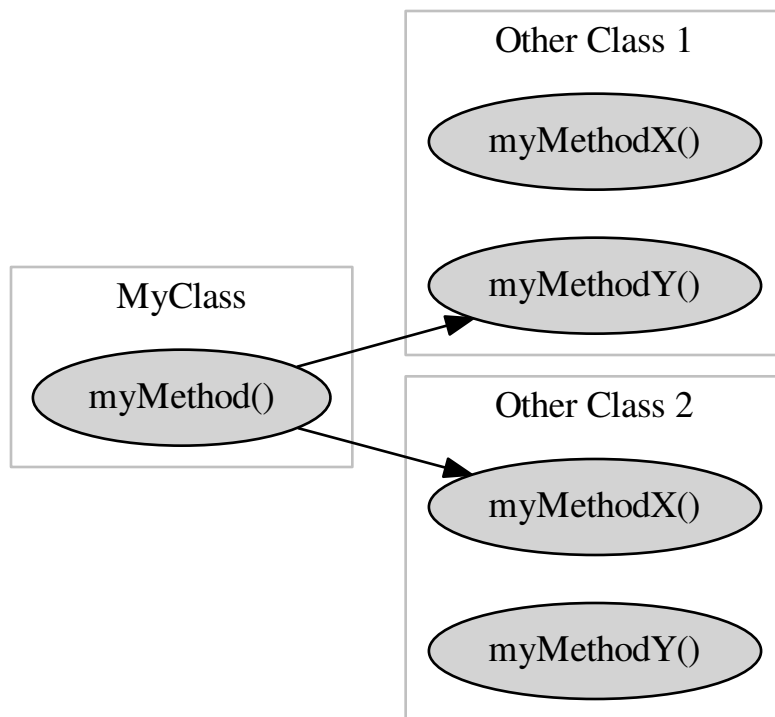


Ok, what about classes/methods that are **invoked by the invoked** classes/methods? :) Well for this purpose there is the **-max-depth** parameter which specifies to which depth the cross calls should be searched. Let's have a look at some examples:

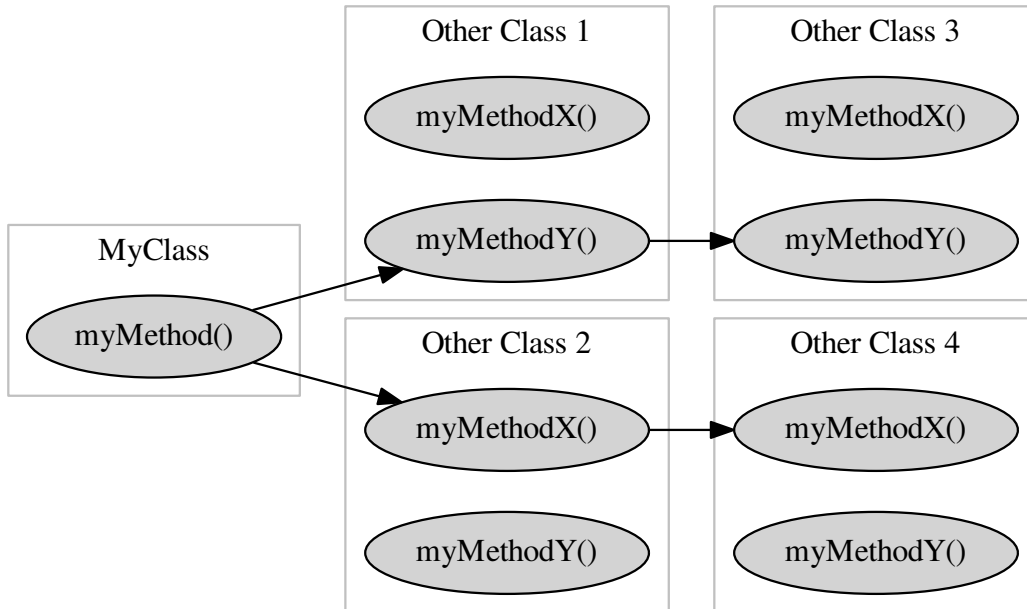
- *-d from -max-depth 0*



- *-d from -max-depth 1*



- *-d from -max-depth 2*



Got it? :)

Examples:

```
smalisca>sxcl -c gmail -m init -d to --max-depth 1
...
smalisca>sxcl -c gmail -m create -d from --max-depth 2
...
```

#### 4.5.4 Web API

In order to improve usability one could use a web API to access previously generated results. At the moment a *REST* like API is available to access:

- classes
- class methods
- class properties
- constant strings
- calls
- cross-calls

Actually you should be able to access every *table* inside your DB using a web client.

## Basic usage

Before being able to access any data make sure you have parsed your Smali files properly and exported the results to a **SQLite** (*-f sqlite*) DB. Afterwards you can start your web service:

```
$ smalisca web --help

...
Usage: smalisca (sub-commands ...) [options ...] {arguments ...}

[--] Analyze results using web API.

optional arguments:
  -h, --help            show this help message and exit
  --debug               toggle debug output
  --quiet               suppress all output
  --log-level {debug,info,warn,error,critical}
                        Change logging level (Default: info)
  -f FILENAME, --file FILENAME
                        Specify SQLite DB (required)
  -H HOST, --host HOST  Specify hostname to listen on
  -p PORT, --port PORT  Specify port to listen on
```

GET /api/<table name>/(int: id)

- you can apply **filters** to the queried results:

GET /api/<table name>?q=<filters>

---

**Note:**

**At the moment only following HTTP methods are allowed:**

- GET
- POST

---

Now let's have a look at some examples:

1. Retrieve *some* classes:

```
$ curl http://localhost:1337/api/classes
```

```
{
  "num_results": 335,
  "objects": [
    {
      "class_name": "Landroid/support/v4/app/BackStackRecord$Op",
      "class_package": "Landroid.support.v4.app",
      "class_type": "final",
      "const_strings": [],
      "depth": 5,
      "id": 1,
      "methods": [
        {
          "id": 1,
          "method_args": "",
          "method_class": "Landroid/support/v4/app/BackStackRecord$Op",
          "method_name": "<init>",
          "method_ret": "V",
          "method_type": "constructor"
        }
      ],
      "path": "/home/victor/tmp/FakeBanker2/dumped/smali/android/support/v4/app/BackStackRecord$Op",
      "properties": [
        {
          "id": 1,
          "property_class": "Landroid/support/v4/app/BackStackRecord$Op",
          "property_info": "",
          "property_name": "fragment",
          "property_type": "Landroid/support/v4/app/Fragment"
        },
        {
          "id": 2,
          "property_class": "Landroid/support/v4/app/BackStackRecord$Op",
          "property_info": "",
          "property_name": "next",
          "property_type": "Landroid/support/v4/app/BackStackRecord$Op"
        },
        {
          "id": 3,
          "property_class": "Landroid/support/v4/app/BackStackRecord$Op",
          "property_info": "",
          "property_name": "next",
          "property_type": "Landroid/support/v4/app/BackStackRecord$Op"
        }
      ]
    }
  ]
}
```

```

        "property_name": "prev",
        "property_type": "Landroid/support/v4/app/BackStackRecord$Op"
    },
    {
        "id": 4,
        "property_class": "Landroid/support/v4/app/BackStackRecord$Op",
        "property_info": "",
        "property_name": "removed",
        "property_type": "Ljava/util/ArrayList"
    }
}

...
"page": 1,
"total_pages": 34
}

```

## 2. Get *class* entry (id = 2):

```
$ curl http://localhost:1337/api/classes/2
```

```

{
  "class_name": "Landroid/support/v4/view/accessibility/AccessibilityRecordCompatIcsMr1",
  "class_package": "Landroid.support.v4.view.accessibility",
  "class_type": "",
  "const_strings": [],
  "depth": 6,
  "id": 2,
  "methods": [
    {
      "id": 2,
      "method_args": "",
      "method_class": "Landroid/support/v4/view/accessibility/AccessibilityRecordCompatIcsMr1",
      "method_name": "<init>",
      "method_ret": "V",
      "method_type": "constructor"
    },
    {
      "id": 3,
      "method_args": "Ljava/lang/Object;",
      "method_class": "Landroid/support/v4/view/accessibility/AccessibilityRecordCompatIcsMr1",
      "method_name": "getMaxScrollX",
      "method_ret": "I",
      "method_type": "public static"
    },
    {
      "id": 4,
      "method_args": "Ljava/lang/Object;",
      "method_class": "Landroid/support/v4/view/accessibility/AccessibilityRecordCompatIcsMr1",
      "method_name": "getMaxScrollY",
      "method_ret": "I",
      "method_type": "public static"
    },
    {
      "id": 5,
      "method_args": "Ljava/lang/Object;I",
      "method_class": "Landroid/support/v4/view/accessibility/AccessibilityRecordCompatIcsMr1",
      "method_name": "setMaxScrollX",
      "method_ret": "V",
      "method_type": "public static"
    }
  ]
}

```

```
    },
    {
      "id": 6,
      "method_args": "Ljava/lang/Object;I",
      "method_class": "Landroid/support/v4/view/accessibility/AccessibilityRecordCompatIcsMr1",
      "method_name": "setMaxScrollY",
      "method_ret": "V",
      "method_type": "public static"
    }
  ],
  "path": "/home/victor/tmp/FakeBanker2/dumped/smali/android/support/v4/view/accessibility/Access
  "properties": []
}%
```

### 3. Get 4-th page of the results:

```
$ curl http://localhost:1337/api/classes?p=4
```

### 4. Apply filters to query results

- Get all classes where class\_name LIKE %android%:

```
$ curl -v -G -H "Content-Type: application/json" \
  -d 'q={"filters":[{"name":"class_name","op":"like","val":"%Creator%"}]}' \
  http://localhost:1337/api/classes

{
  "num_results": 4,
  "objects": [
    {
      "class_name": "Landroid/support/v4/os/ParcelableCompatCreatorCallbacks",
      "class_package": "Landroid.support.v4.os",
      "class_type": "public interface abstract",
      "const_strings": [],
      "depth": 5,
      "id": 10,
      "methods": [
        {
          "id": 89,
          "method_args": "Landroid/os/Parcel;Ljava/lang/ClassLoader;",
          "method_class": "Landroid/support/v4/os/ParcelableCompatCreatorCallbacks",
          "method_name": "createFromParcel",
          "method_ret": "Ljava/lang/Object;",
          "method_type": "public abstract"
        },
        {
          "id": 90,
          "method_args": "I",
          "method_class": "Landroid/support/v4/os/ParcelableCompatCreatorCallbacks",
          "method_name": "newArray",
          "method_ret": "[Ljava/lang/Object;",
          "method_type": "public abstract"
        }
      ],
      "path": "/home/victor/tmp/FakeBanker2/dumped/smali/android/support/v4/os/ParcelableCom
      "properties": []
    },
    ...
  ]
}
```



```
}
```

- Get all classes where id > 10:

```
$ curl -v -G -H "Content-Type: application/json" \
  -d 'q={"filters":[{"name":"id","op":"ge","val":10}]}' \
  http://localhost:1337/api/classes
```

```
...
```

- Get all classes where id > 10 AND class\_type like “%final%”:

```
$ curl -v -G -H "Content-Type: application/json" \
  -d 'q={"filters":[{"and":[{"name":"id","op":"ge","val":10}, {"name":"class_type","op":
```

```
...
```

---

**Note:** For additional examples make sure you have a look at [Making search queries](#) inside the Flask-Restless documentation.

---

## 4.5.5 Draw results

### Basic usage

All **drawing** commands start with a **d**. The d-commands will first invoke a s-command (search) to get the results and generate a graph afterwards. You can specify different output formats (by -f) which best fit to your needs. A required output file name (-o) also has to be specified.

---

**Note:** Don't miss the *screenshots*.

---

### Styling

The layout styles are available via a config file (*-config*). If you don't specify any config file then the default one will be used: *smalisca/data/config/config.conf*.

### Available commands

#### dc

[D]raws [c]lass graphs. Similar to the *sc* command you can search for classes by specifying the column type (-c) and a pattern (-p). General usage:

```
smalisca>dc --help
usage: dc [-h] [-c SEARCH_TYPE] [-p SEARCH_PATTERN]
          [-f {dot,xdot,png,pdf,jpg,svg}]
          [--prog {dot,neato,circo,twopi,fdp,sfdp,nop}] [--args OUTPUT_ARGS]
          -o OUTPUT
```

```
>> Draw class graphs
```

optional arguments:

```
-h, --help            show this help message and exit
-c SEARCH_TYPE        Specify column.
                        Type ? for list
-p SEARCH_PATTERN     Specify search pattern
-f {dot,xdot,png,pdf,jpg,svg}
                        Output format
                        Default: dot
--prog {dot,neato,circo,twopi,fdp,sfdp,nop}
                        Graphviz layout method
                        Default: dot
--args OUTPUT_ARGS    Additional graphviz arguments
-o OUTPUT            Specify output file
```

#### Examples:

```
smalisca>dc -c class_name -p com/gmail/xservices -f dot -o /tmp/calls.dot
:: INFO           Wrote results to /tmp/calls.dot
```

#### You can of course use other **output formats**:

```
smalisca>dc -c class_name -p com/gmail/xservices -f png -o /tmp/calls.png
:: INFO           Wrote results to /tmp/calls.png
```

#### Or a different **graphviz engine**:

```
smalisca>dc -c class_name -p com/gmail/xservices -f png --prog fdp -o /tmp/calls.png
:: INFO           Wrote results to /tmp/calls.png
```

### **dcl**

[D]raws calls [cl]. Similar to the *sc/* command you can search for calls by specifying the calling method/class and/or the destined method/class. General usage:

```
smalisca>dcl --help
usage: dcl [-h] [-fc FROM_CLASS] [-fm FROM_METHOD] [-tc TO_CLASS]
          [-tm TO_METHOD] [-fa LOCAL_ARGS] [-ta DEST_ARGS]
          [-f {dot,xdot,png,pdf,jpg,svg}]
          [--prog {dot,neato,circo,twopi,fdp,sfdp,nop}] [--args OUTPUT_ARGS]
          -o OUTPUT
```

>> Draw calls graphs

optional arguments:

```
-h, --help            show this help message and exit
-fc FROM_CLASS        Specify calling class (from)
-fm FROM_METHOD       Specify calling method (from)
-tc TO_CLASS          Specify destination class (to)
-tm TO_METHOD         Specify destination method (to)
-fa LOCAL_ARGS        Local arguments (from)
-ta DEST_ARGS         Destination arguments (to)
-f {dot,xdot,png,pdf,jpg,svg}
                        Output format
                        Default: dot
--prog {dot,neato,circo,twopi,fdp,sfdp,nop}
                        Graphviz layout method
                        Default: dot
--args OUTPUT_ARGS    Additional graphviz arguments
-o OUTPUT            Specify output file
```

Let's have a look at some examples:

```
smalisca>dcl -fc gmail -tm create -f pdf --prog fdp -o /tmp/smalisca/calls.pdf
:: INFO      Wrote results to /tmp/smalisca/calls.pdf
```

## dxcl

[D]raws cross [x] calls [cl]. Similar to the *sxcl* command you can search for cross-calls by specifying class name and/or method name. General usage:

```
smalisca>dxcl --help
usage: dxcl [-h] [-c CLASS_NAME] [-m METHOD_NAME] -d {to,from}
           [--max-depth [XREF_DEPTH]] [-f {dot,xdot,png,pdf,jpg,svg}]
           [--prog {dot,neato,circo,twopi,fdp,sfdp,nop}] [--args OUTPUT_ARGS]
           -o OUTPUT
```

>> Draw cross-calls graphs

optional arguments:

```
-h, --help          show this help message and exit
-c CLASS_NAME       Specify class name
-m METHOD_NAME       Specify method name
-d {to,from}        Cross-reference direction
--max-depth [XREF_DEPTH]
                    Cross-References max depth
                    Default: 1
-f {dot,xdot,png,pdf,jpg,svg}
                    Output format
                    Default: dot
--prog {dot,neato,circo,twopi,fdp,sfdp,nop}
                    Graphviz layout method
                    Default: dot
--args OUTPUT_ARGS  Additional graphviz arguments
-o OUTPUT           Specify output file
```

Let's have a look at some examples:

```
smalisca>dxcl -c gmail/xlibs -d to --max-depth 1 -f pdf --prog dot -o /tmp/smalisca/xcalls.pdf
:: INFO      Namespace(to_class='gmail/xlibs')
:: INFO      Wrote results to /tmp/smalisca/xcalls.pdf
```

## 4.5.6 Screenshots

### General usage


```
$ smalisca --help
```

```
$ smalisca parser -l ~/tmp/FakeBanker2/dumped/smali -s java -f sqlite -o /tmp/fakebanker.sqlite
```

```

-> smalisca --help

```



```

:: Author:      Victor <Cyneox> Dorneanu
:: Desc:        Static Code Analysis tool for Smali files
:: URL:         http://nullsecurity.net, http://(blog,www).dornea.nu
:: Version:     0.1

```

```

usage: smalisca (sub-commands ...) [options ...] [arguments ...]

[--] Static Code Analysis (SCA) tool for Baskmali (Smali) files.

commands:

  analyzer
    [--] Analyze results using an interactive prompt or on the command line.

  parser
    [--] Parse files and extract data based on Smali syntax.


optional arguments:
  -h, --help            show this help message and exit
  --debug               toggle debug output
  --quiet               suppress all output
  --log-level {debug,info,warn,error,critical}
                        Change logging level (Default: info)
  -v, --version          show program's version number and exit

```

```

-> smalisca parser -t ~/tmp/FakeBanker2/dumped/smali -s java -f sqlite -o ~/tmp/fakebanker.sqlite

```



```

:: Author:      Victor <Cyneox> Dorneanu
:: Desc:        Static Code Analysis tool for Smali files
:: URL:         http://nullsecurity.net, http://(blog,www).dornea.nu
:: Version:     0.1

```

```


:: INFO      Parsing .java files in /home/victor/tmp/FakeBanker2/dumped/smali ...
:: INFO      Finished parsing!
:: INFO      Exporting results to SQLite
:: INFO      Extract classes ...
:: INFO      Extract class properties ...
:: INFO      Extract class methods ...
:: INFO      Extract calls ...
:: INFO      Commit changes to SQLite DB
:: INFO      Wrote results to ~/tmp/fakebanker.sqlite
:: INFO      Finished scanning

```

```

-> ./smalisca.py analyzer -i neu.sqlite -f sqlite

```



```

:: Author:      Victor <Cyneox> Dorneanu
:: Desc:        Static Code Analysis tool for Smali files
:: URL:         http://nullsecurity.net, http://(blog,www).dornea.nu
:: Version:     0.1

```

```

:: INFO      Successfully opened SQLite DB
:: INFO      Creating analyzer framework ...
:: INFO      Starting new analysis shell

```

```

-- Analyzer
Welcome to smalisca analyzer shell.
Type ? or help to list available commands.
Type "<command> --help" for additional help.

smalisca>sc -c class_name -p com/gmail/xpack -r 10

```

id	class_name	class_type	class_package	depth	path
1	Lcom/gmail/xpack/R\$style	public final	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/R\$style.java
3	Lcom/gmail/xpack/ActUpdate	public	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/ActUpdate.java
89	Lcom/gmail/xpack/R\$raw	public final	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/R\$raw.java
101	Lcom/gmail/xpack/MainActivity\$1	public final	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/MainActivity\$1.java
161	Lcom/gmail/xpack/R	public final	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/R.java
168	Lcom/gmail/xpack/BuildConfig	public final	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/BuildConfig.java
245	Lcom/gmail/xpack/R\$id	public final	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/R\$id.java
268	Lcom/gmail/xpack/R\$drawable	public final	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/R\$drawable.java
273	Lcom/gmail/xpack/R\$string	public final	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/R\$string.java
299	Lcom/gmail/xpack/R\$layout	public final	Lcom.gmail.xpack	4	/home/victor/tmp/FakeBanker2/dumped/smali/com/gmail/xpack/R\$layout.java

```

smalisca>_

```

Figure 4.1: Search for classes using interactive prompt

```
smalisca>scl -fc com/gmail/xpack -x local_args,dst_args -r 30 -s dst_method
```

id	from_class	from_method	dst_class	dst_method	ret
1	Lcom/gmail/xpack/R\$style	<init>	Ljava/lang/Object	<init>	V
10	Lcom/gmail/xpack/ActUpdate	doUpdate	Landroid/content/Intent	<init>	V
1742	Lcom/gmail/xpack/R\$raw	<init>	Ljava/lang/Object	<init>	V
1811	Lcom/gmail/xpack/MainActivity\$1	<init>	Ljava/lang/Object	<init>	V
2500	Lcom/gmail/xpack/R	<init>	Ljava/lang/Object	<init>	V
2824	Lcom/gmail/xpack/BuildConfig	<init>	Ljava/lang/Object	<init>	V
5077	Lcom/gmail/xpack/R\$id	<init>	Ljava/lang/Object	<init>	V
5798	Lcom/gmail/xpack/R\$drawable	<init>	Ljava/lang/Object	<init>	V
5832	Lcom/gmail/xpack/R\$string	<init>	Ljava/lang/Object	<init>	V
5896	Lcom/gmail/xpack/R\$layout	<init>	Ljava/lang/Object	<init>	V
5922	Lcom/gmail/xpack/R\$attr	<init>	Ljava/lang/Object	<init>	V
6005	Lcom/gmail/xpack/R\$menu	<init>	Ljava/lang/Object	<init>	V
6239	Lcom/gmail/xpack/MainActivity	<init>	Landroid/app/Activity	<init>	V
6240	Lcom/gmail/xpack/MainActivity	<init>	Lcom/gmail/xpack/MainActivity\$1	<init>	V
8	Lcom/gmail/xpack/ActUpdate	<init>	Landroid/app/Activity	<init>	V
6241	Lcom/gmail/xpack/MainActivity	doCheckPass	Lcom/gmail/xpack/MainActivity	findViewById	Landroid/view/View;
6242	Lcom/gmail/xpack/MainActivity	doCheckPass	Lcom/gmail/xpack/MainActivity	findViewById	Landroid/view/View;
9	Lcom/gmail/xpack/ActUpdate	doUpdate	Landroid/net/Uri	fromParts	Landroid/net/Uri;
6243	Lcom/gmail/xpack/MainActivity	doCheckPass	Landroid/widget/EditText	getText	Landroid/text/Editable;
6246	Lcom/gmail/xpack/MainActivity	doCheckPass	Landroid/widget/EditText	getText	Landroid/text/Editable;
6249	Lcom/gmail/xpack/MainActivity	doCheckPass	Ljava/lang/String	length	I
6250	Lcom/gmail/xpack/MainActivity	doCheckPass	Ljava/lang/String	length	I
12	Lcom/gmail/xpack/ActUpdate	onCreate	Landroid/app/Activity	onCreate	V
13	Lcom/gmail/xpack/ActUpdate	onCreate	Lcom/gmail/xpack/ActUpdate	setContentView	V
6251	Lcom/gmail/xpack/MainActivity	doCheckPass	Lcom/gmail/xpack/MainActivity	showDialog	V
11	Lcom/gmail/xpack/ActUpdate	doUpdate	Lcom/gmail/xpack/ActUpdate	startActivity	V
6244	Lcom/gmail/xpack/MainActivity	doCheckPass	Landroid/text/Editable	toString	Ljava/lang/String;
6247	Lcom/gmail/xpack/MainActivity	doCheckPass	Landroid/text/Editable	toString	Ljava/lang/String;
6245	Lcom/gmail/xpack/MainActivity	doCheckPass	Ljava/lang/String	trim	Ljava/lang/String;
6248	Lcom/gmail/xpack/MainActivity	doCheckPass	Ljava/lang/String	trim	Ljava/lang/String;

Figure 4.2: Search for calls using interactive prompt

```
smalisca>help
Documented commands (type help <topic>):
-----
dc dcl dxcl help q quit sc scl sm sp sxcl

smalisca>help sxcl
Search for cross calls. Type 'sxcl --help' for help.
smalisca>sxcl --help
usage: sxcl [-h] [-c CLASS_NAME] [-m METHOD_NAME] -d {to,from}
           [--max-depth [XREF_DEPTH]] [-s SORTBY] [--reverse] [-r RANGE]
           [--max-width MAX_WIDTH] [-x EXCLUDE_FIELDS]

>> Search for calls

You can apply filters by using the optional arguments.
Without any arguments the whole 'calls' table will
be printed.

optional arguments:
  -h, --help            show this help message and exit
  -c CLASS_NAME          Specify class name
  -m METHOD_NAME          Specify method name
  -d {to,from}           Cross-reference direction
  --max-depth [XREF_DEPTH]
                        Cross-References max depth
                        Default: 1
  -s SORTBY              Sort by column name
  --reverse              Reverse sort order
  -r RANGE               specify output range by single integer or separated by ','
  --max-width MAX_WIDTH
                        Global column max width
  -x EXCLUDE_FIELDS      Exclude table fields
```

Figure 4.3: General help / usage

## Analysis

## Drawing

### dc

```
smalisca>dc -c class_name -p gmail/xlibs -f png --prog dot -o /tmp/smalisca/classes.png
```

```
smalisca>dc -c class_name -p gmail/xlibs -f png --prog fdp -o /tmp/smalisca/classes.png
```

### dcl

```
smalisca>dcl -fc gmail/xlibs -fm init -f png --prog dot -o /tmp/smalisca/calls.png
```

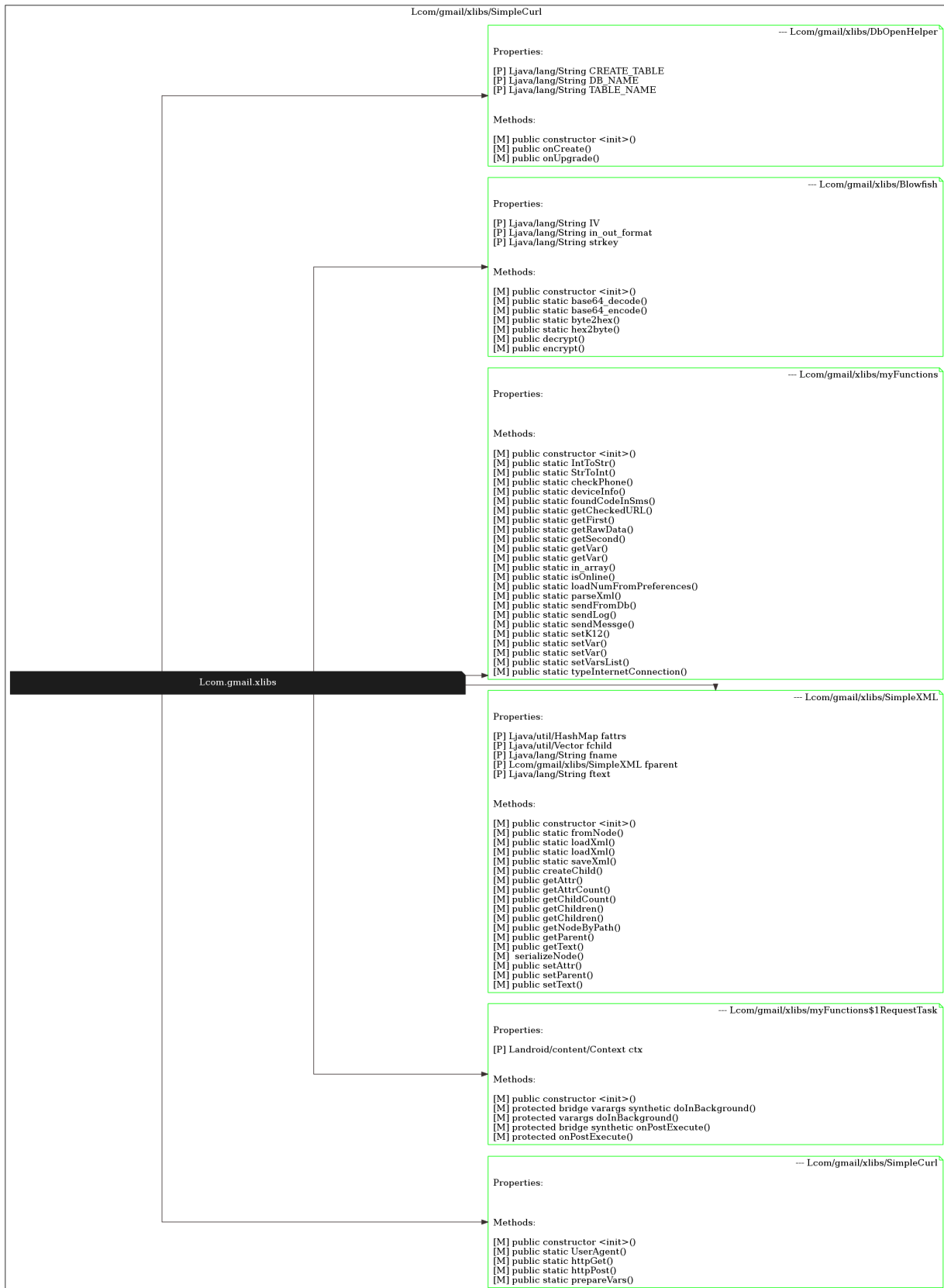
```
smalisca>dcl -fc gmail/xlibs -fm init -f png --prog fdp -o /tmp/smalisca/calls.png
```

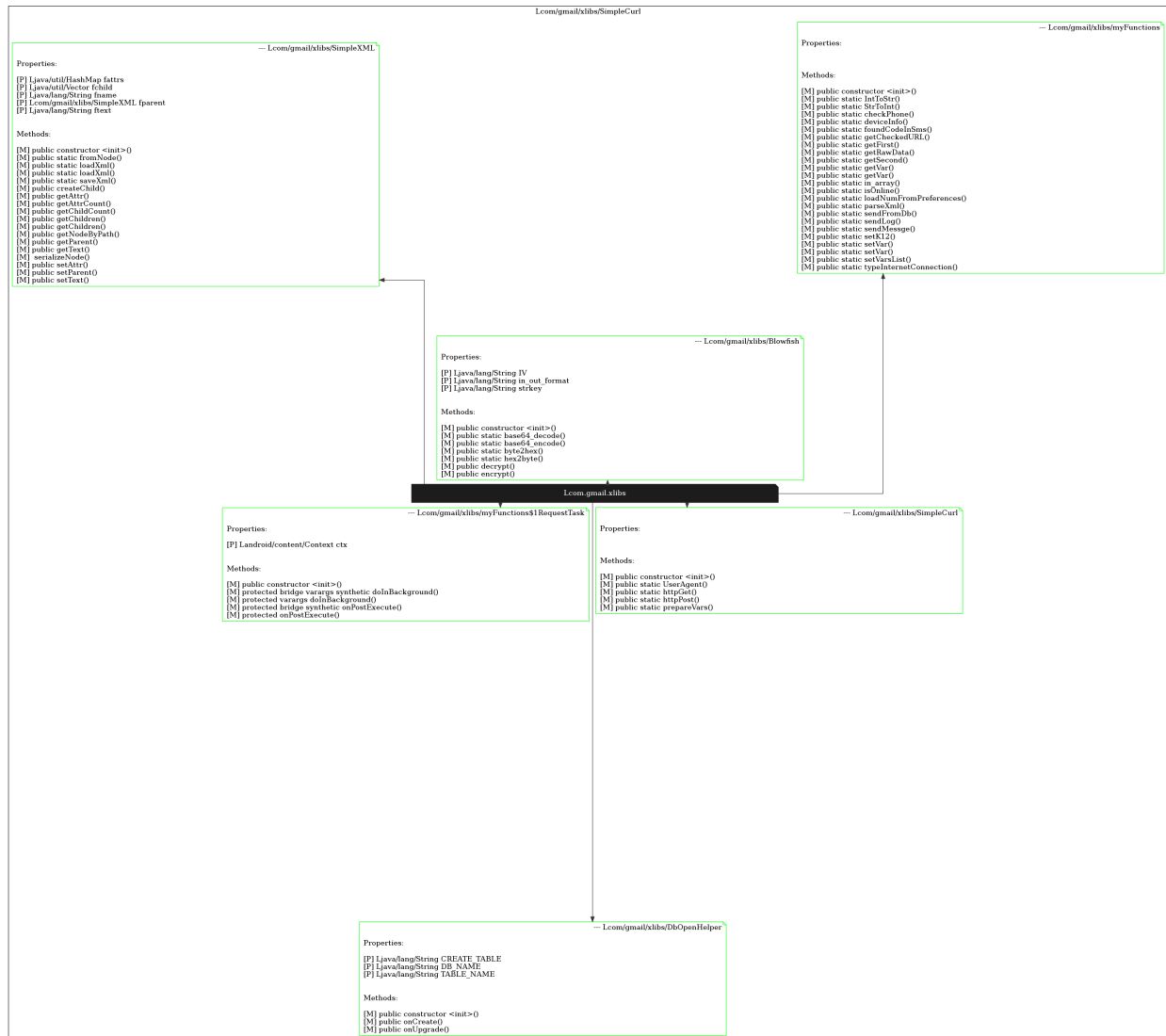
### dxcl

```
smalisca>dxcl -c gmail/xlibs -d to --max-depth 0 -f png --prog dot -o /tmp/smalisca/xcalls.png
```

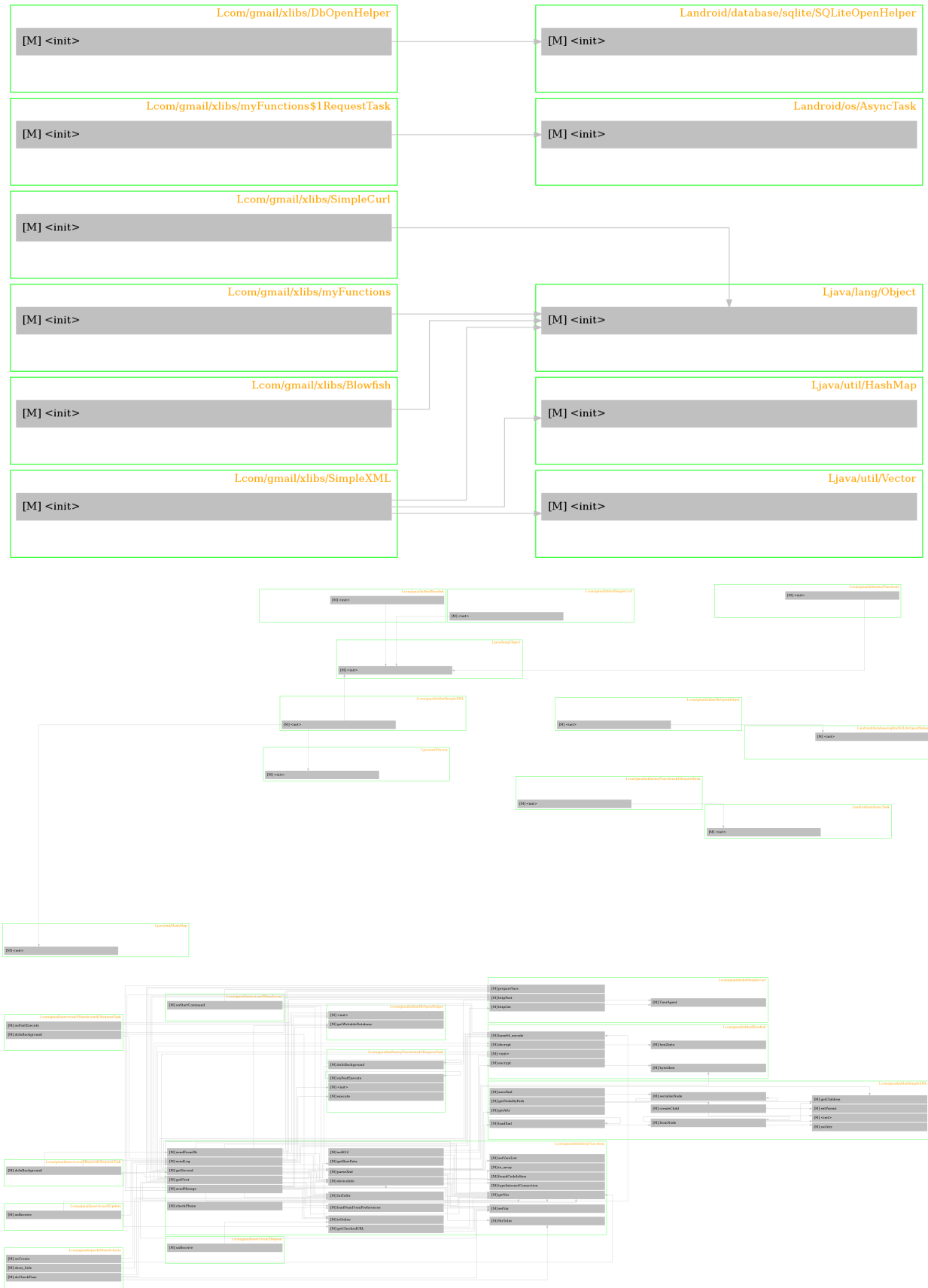
```
smalisca>dxcl -c gmail/xlibs -d to --max-depth 1 -f png --prog dot -o /tmp/smalisca/xcalls.png
```

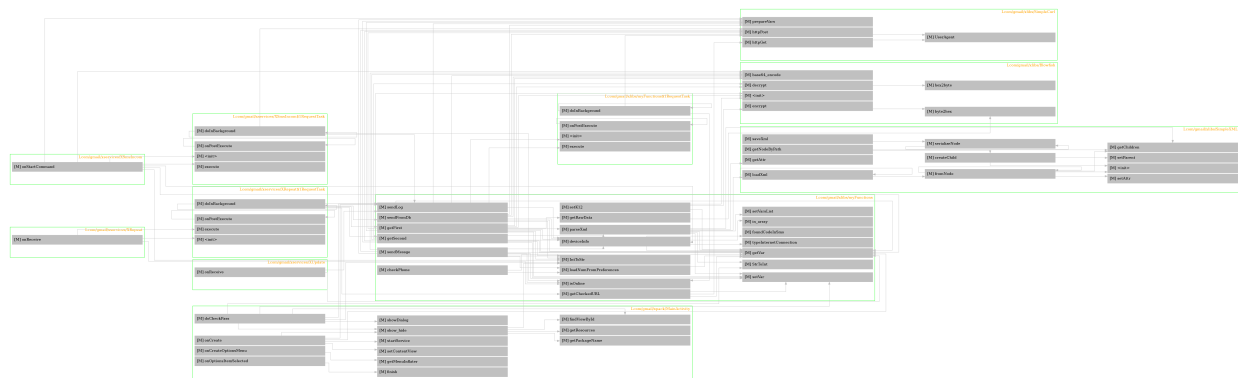
```
smalisca>dxcl -c gmail -m create -d from --max-depth 1 -f png --prog dot -o /tmp/smalisca/xcalls.png
```

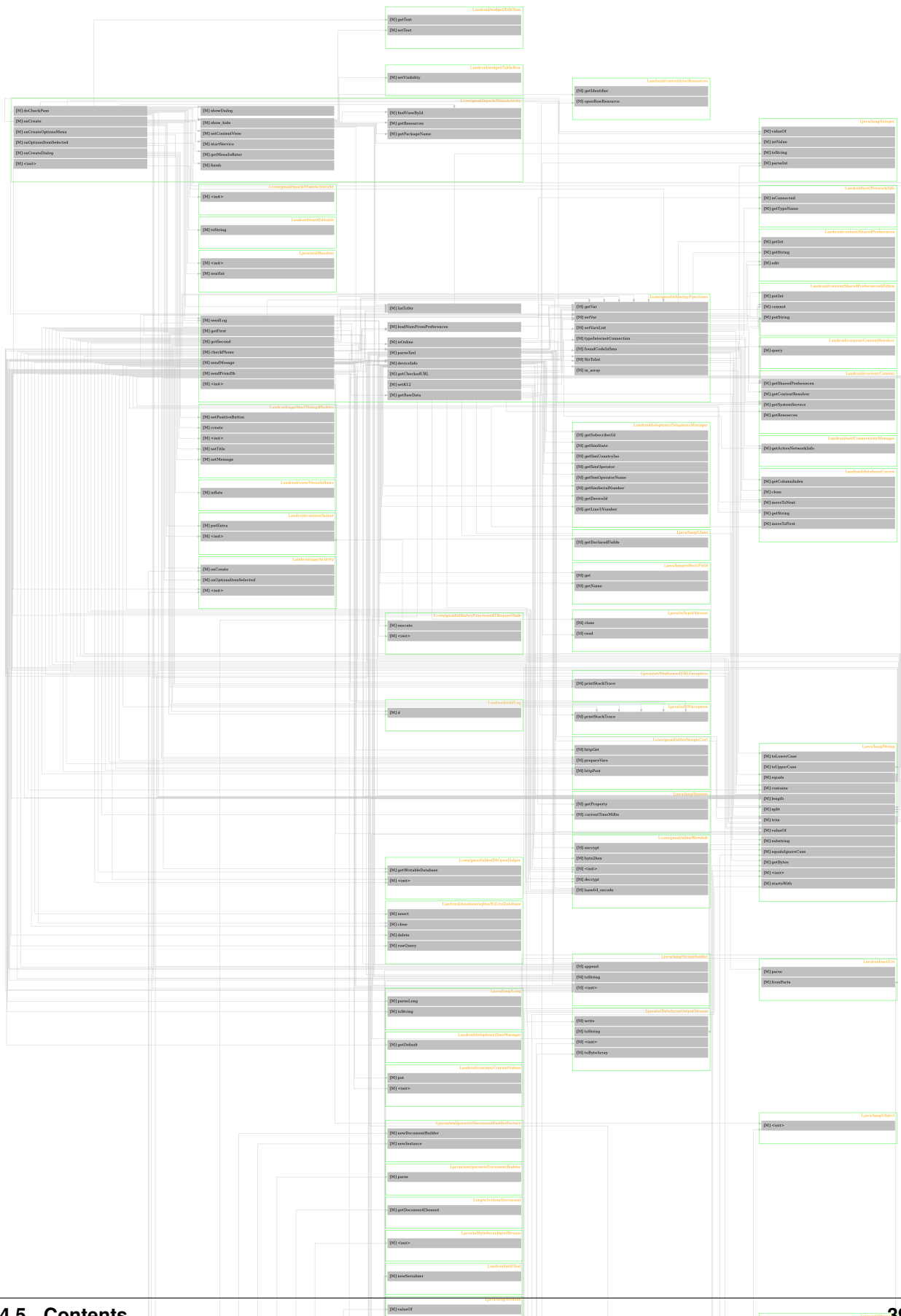














---

## Indices and tables

---

- *genindex*
- *modindex*
- *search*