Caixa Mágica

Aptoide client / server interfaces

Date:	*1/27/11*
Pages:	*18*
Issue:	*Aptoide - client / server interfaces*
State:	*Working Document*
Access:	*Public*
Reference:	*Aptoide - client / server interfaces*

Approved Version: RR			
Name	**Function**	**Signature**	**Date**

Authors and Contributors:			
Name	**Contact**	**Description**	**Date**
Roberto Jacinto	Roberto.Jacinto@caixamagica.pt	Developer	22-10-09
Pedro Fragoso	Pedro.Fragoso@caixamagica.pt	Developer	22-10-09
Paulo Trezentos	Paulo.Trezentos@caixamagica.pt	Technical director	22-10-09

Access List:
Internal Access
External Access

Revision History:			
Version	**Date**	**Description**	**Author**
0.5	15-10-09	First draft of the document	RJ
0.51	22-10-09	Revision	PT
0.52	23-10-09	Revision	RJ
0.6	10-02-10	Revision	RJ
0.7	04-03-10	Revision	RJ
1.2	01-04-10	Revision	RJ
2.2	25-01-11	Revision and update	RJ
2.4	23-02-11	Document structure revision. Added Qrcode and direct link xml interfaces.	RJ

Index

[AD 1]Introduction

In this document, we are going to give an overview of all the interfaces accepted by *Aptoide client,* whether to interact directly with a remote server (repository), or to interact with *client* in a more advanced way.

We will start by presenting the basic access functions for a working interaction between a repository server and the client – the **info.xml** and **extra.xml** files – providing all the information and some examples. After, we'll show how the basic interactions can be upgraded to provide a more pleasant user experience.

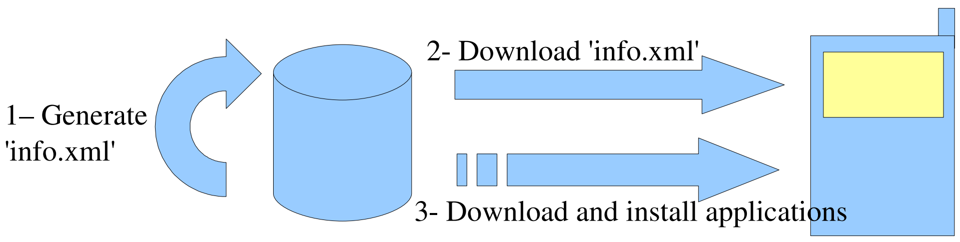
[AD 2]Basic access to a repository

The major communication in *Aptoide* project, is between the client and the remote repositories. Every info from every package present in a given repository must be sent to the client side, as fast and as accurate possible.

This channel of communication is based on the '**info.xml**' file, which must be present in every repository; a simple XML file with a number of tags used in accordance to the packages provided.

Let's first star by understanding how *Aptoide* uses '**info.xml**' file.

[AD 1]Basic working mechanism



The diagram above, shows how an interaction between a repository and the client would normally behave.

First, we must generate the XML file with all the information about the applications in the repository. Next, we publish that XML file – **info.xml** – to be retrieved by any *Aptoide* device. Finlay, *Aptoide* uses the information it gathered to download and install applications.

[AD 2]Repository configurations

All you really need to create an *Aptoide* based repository is a server running some web server – like apache.

The information needed for a basic interaction[1] is all present in the info.xml file presented above. Apart from the possibility to access that file, you will obviously need to gave the user access to the application files – apk – themselves.

For better understanding, we are going to use a imaginary repository. Let's say we have a web accessible server, running apache, with the Internet address "http://my.repo.com".

Accessing that address, will put us in our main server directory, in this example htdocs.

[1] Basic interaction meaning the capability of listing all applications on the server, and the possibility to install them.

Now, to create a *Aptoide* repository, you just need to put the **info.xml** file in the directory you will give away as your repository address.

If we create the directory 'arepo' and place it in our main directory – *htdocs* – we would put info.xml in there. The address that should be supplied to access your repository would be: 'http://my.repo.com/arepo'.

[AD 3]Generating the XML file

There are a number of ways one might create the **info.xml** file. From asking directly to the user o supplies the application, to more scripted ways, were information is gathered from the *.apk file itself.

Although we supply a basic generator, the scope of this document is to present all the informations about the interfaces themselves, and not how one can more easily or not generate them.

For now, the only thing you need to know, is that there is a info.xml file, witch is the basic channel for *Aptoide* to communicate with a repository server. All repositories must have one, and the structure of it depends directly on the application file you wish to distribute, and the information about them, you wish to make available.

[AD 4]Downloading the XML file

The first thing *Aptoide* must do to use a new repository, is to retrieve the **info.xml** file from it.

All the information needed to download and install applications is there.

As said above, the file must be in the same directory that the Internet address of it access.

After downloading the file, *Aptoide* will parse all the information on it, and generate a list with all the applications information. Be aware that if an application have an icon, *Aptoide* will connect to the repository again to retrieve it.

[AD 5]Downloading applications

When a user tries to download an application, *Aptoide* tries to get it based on the information included in the XML file.

The applications can be anywhere as long as they are inside the same repository – in other words, they can be in a different directory from the **info.xml** file, but they must be accessible from the same entry point where the xml file was.

[AD 3]Creating your repository

As we showed above, to create your one repository, you need an accessible web server to which the user can access through the web, the apk files you want to make available, and the basic file that *Aptoide* client accesses in order to populate itself: the **info.xml** file.

[AD 1] The info.xml file

This XML file, as most, starts with the standard informations header. Just a basic one is needed as *Aptoide* don't parse any of this information.
Next, the opening tag is used: **<apklst>**. This is the enclosed tag for the entire file.
Next we must delimit every application. We use the tag **<package>** for that.
As an example, imagine we have two applications in the imaginary repository X, the 'info.xml' file must have the below skeleton:

```
<?xml version="1.0" encoding="UTF-8"?>
<apklst>
  <package>
    /* all information about application 1 goes here*/
  </package>
  <package>
    /* all information about application 2 goes here*/
  </package>
</apklst>
```

[AD 2] Mandatory tags – info.xml

The only information *Aptoide* needs to function is the application package - which is unique to every application, and therefor identifies the application – and the path to the application *.apk in the repository:

- **Application package:** This string identifies the application and is unique. It's enclosing tag is **<apkid>**.
- **Application file path:** This is the relative path where the *.apk file for this application is in the remote repository. It's enclosing tag is **<path>**.

[AD 3] Optional tags – info.xml

This tags are used to pass additional information about applications. Although they are not mandatory – *Aptoide* can operate with just the tags presented above – they are very useful for improving the user experience while operating *Aptoide*.

- **Application name:** The truth is, *Aptoide* don't really need to know the application name. A application is identified by it's package name. Even sow, it's simpler to the user to have a more human-readable way to identify them. The enclosing tag for this property is **<name>**. When no value is passed, *Aptoide* presents the package name as the application name.

- **Version name:** The version of the application present in the repository is useful for user information. If no version is specified, *Aptoide* declares it as version 0.0. The enclosing tag is **<ver>**.
- **Version code:** This is the application version code. Although it is present in every application, it is not a mandatory tag. It is however advised to use it, as it is by it that *Aptoide* controls upgrades. The default value for it is 0, and the enclosing tag is **<vercode>**.
- **Application icon:** Not every application has an icon. If it has, the path to the icon file in the server must be passed. If there is no icon, or that information is not passed, *Aptoide* uses the standard Android icon. The enclosing tag is **<icon>**.
- **Category:** Applications can be joined in categories. For example, an application can be a game, an office application or other things. This information is used to modify the way the list of applications appear on *Aptoide*. If no category information is available, it will be put on "other" category. This information has two levels on Aptoide. Level one is either an *Application*, a *Game* or *Other.* This information goes on the **<catg>** tag. Second level information (you can check a list of supported ones in the end of this document) is passed on the **<catg2>** tag.
- **Date:** As category, date refers to the date where the application first became available in the repository. In the same manner as above, this information might be used for user interface only, were the user might want to display applications in a "most recent first" order. The enclosing tag is **<date>**.
- **Download number:** Indicates the number of downloads this apk file as had from the server it refers to. The enclosing tag is **<dwn>.**
- **Md5 hash:** The md5 hash of an application apk file is used to guarantee that the file you download from the server is the same that you are going to install. The tag for this is **<md5h>.**

[AD 4]XML file example – info.xml

For better perception we supply an overview example.
Let's say we have the repository 'http://aptoide.repo.com". In there, we have two applications, "App1" and "App2". We also have a directory called "icons" where icons are stored.
General applications information:

	Application 1	**Application 2**
Package	ex.app2.com	ex.app2.com
Path	App1.apk	App2.apk
Name	- - -	App 2
Version	- - -	'2.1.1
Icon	- - -	/icons/app2.ico
Main Category	- - -	Games
Second Category	- - -	Cards & Casino
Date	- - -	21-01-10
MD5	- - -	30-12-99

As shown in the table above, application 1 only have the basic information available. Has for application 2, there is every thing we might want to know. The 'info.xml' file for this repository would be:

```
<?xml version="1.0" encoding="UTF-8"?>
<apklst>
  <package>
        <apkid>ex.app1.com</apkid>
        <path>App1.apk</path>
  </package>
  <package>
        <apkid>ex.app2.com</apkid>
        <path>App2.apk</path>
        <name>App 2</name>
        <ver>2.1.1</ver>
        <icon>/icons/app2.ico</icon>
        <catg>Games</catg>
        <catg2>Cards & Casino</catg2>
        <date>21-01-10</date>
        <md5h>Games</md5h>
  </package>
</apklst>
```

In this case, *Aptoide* client will present both applications correctly, adding default information when optional tags are not present – default values can be found in the end of this document, on the “quick reference tables”.

[AD 5]Extras XML file

Apart from the **info.xml** file, the *Aptoide* client version 1.2 and above, also supports the **extras.xml** file – placed in the same directory as the **info.xml**.
This new xml file is used to provide a small description of the applications present in the repository, and my or might not be present.
This file can't be automatically generated from the application apk file, and must be created by the repository administrator.

[AD 6]XML tags overview – extras.xml

The XML file, as most, starts with the standard informations header. Just a basic one is needed as *Aptoide* don't parse any of this information.
Next, the opening tag is used: **<extras>**. This is the enclosed tag for the entire file.
Next we must delimit every application we wish to add a description. We use the tag **<pkg>** for that.
As an example, imagine we have two applications in the imaginary repository X, the 'extras.xml' file must have the below skeleton:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<extras>
  <pkg>
    /* the extra information of application 1 goes here*/
  </pkg>
  <pkg>
    /* the extra information of application 2 goes here*/
  </pkg>
</extras>
```

[AD 7]Mandatory tags – extras.xml

- **Application package:** This string identifies the application and is unique. It's enclosing tag is **<apkid>**.

[AD 8]Optional tags – extras.xml

- **Application description:** This string gives a small description of the application. The description can be as long as you wish, however for user interaction and database space, you should try and keep it below 512 characters. It's enclosing tag is **<cmt>**.

Note that, although this tag is optional, the non existence of it will make the xml file for no use, as no information will be parsed from it.

[AD 9]XML file example – extras.xml

For better perception we supply an overview example.
Let's say we have the repository 'http://aptoide.repo.com”. In there, we have two applications, “App1” and “App2”. We already have the **info.xml** file, and wish to provide some description about application 2.
General applications information:

	Application 1	**Application 2**
Package	- - -	ex.app2.com
Description	- - -	This is the version 1.2 of the application 2. Now with network support!

As shown in the table above, we don't want to provide any infomration about application 1 . Has for application 2, we want to add the description presented.

```
<?xml version="1.0" encoding="UTF-8"?>
<extras>
  /* we could omit this entry, as no information about app 1 will be given */
  <pkg>
        <apkid>ex.app1.com</apkid>
        <cmt></cmt>
  </pkg>
  <pkg>
        <apkid>ex.app2.com</apkid>
        <cmt>App2.apk</cmt>
  </pkg>
</extras>
```

[AD 4] Other interfaces

Besides the XML files described above, there are other inputs *Aptoide client* can handle. Those are available, not for the *Aptoide* to communicate with a repository server, but for users to directly command the client to an action.

[AD 1] Adding repositories

Although you can write the repository link directly in the client application, there are other ways, namely by direct link or a Qrcode.

Adding by URL link

To achieve this, *Aptoide* is accepting a special mime-type associated to a controlled crafted xml file.
The mime-type is **vnd.cm.aptoide.pt.** and any download, made inside you device running *Aptoide*, that as this specific mime-type will open the client to parse whatever information comes. In our case, to add a repository link, we will pass a XML file containing the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<newserver>
        <server> … </server>
              …
        <server> … </server>
</newserver>
```

There can be any number of <server> tags, hich one containing a repository URL that is to be added to *Aptoide.*

As an example, if you want to distribute three repositories – http://repo1.org, http://repo2.org and http://repo3.org – in one single file, you would create a xml file, associate it the mime-type above, and made it available for download to everyone. The xml file will look something like:

```
<?xml version="1.0" encoding="UTF-8"?>
<newserver>
        <server> http://repo1.org</server>
        <server> http://repo2.org</server>
        <server> http://repo3.org</server>
</newserver>
```

NOTE: The mime-type functionality described above is supported by all Aptoide clients with version above 1.6.

Adding by Qrcode

Another approach supported by *Aptoide* is qrcodes. Qrcodes are "specific matrix barcode (or two-dimensional code), readable by dedicated QR barcode readers and camera phones".[2]

They present a fast and easy way to add repositories since they can be placed almost anywhere, giving the user a fast way to add them.

In our case, we use the start protocol tag: **aptoiderepo.**
This is supported on *Aptoide* client since 2.0.2. It enables you to create a Qrcode for adding a new repository. All you need to do is generate the following: **aptoiderepo://<your repository here>.** Then generate de Qrcode for the string and it will be readable by any barcode reader and automatically lunch *Aptoide* after parsing.

As an example, pretend you have the repository: http://myrepo.my. Create the corresponding URI: aptoiderepo://http://myrepo.my and generate the corresponding Qrcode. For this particular example, the image generated would be:

[AD 2] Direct link downloads

Another way to interact with the *Aptoide* client, is to give him a direct link to an application file.
You might want to do this when you are having problems with your repository or there are a large number of connections updating the repository list.

[2] Wikipedia: http://en.wikipedia.org/wiki/QR_Code

For that – and since version 2.0.2 - the mime-type xml model have been extended to support this. For that, the following tags were added:

- **<myapp>:** Becomes the root tag for the xml file;
- **<getapp>:** The root tag for the application download functionality;
- **<name>:** The name of the application we wish to fetch;
- **<get>:** The direct link to the apk file.

This ways, a complete overview of the xml file used with the mime-type approach is as follow:

```
<?xml version="1.0" encoding="UTF-8"?>
<getapp>
     <name> … </name>
     <get> … </get>
</getapp>
<newserver>
     <server> … </server>
               ...
     <server> … </server>
</newserver>
```

Note that you can have any number of servers and applications in one file.

NOTE: The mime-type functionality described above is supported by all Aptoide clients with version above 2.0.2.

Since version 2.4, *Aptoide* have also implemented the XML file presented above, and use it on Qrcode. The rules are exactly the same with the exception of the mime-type. There is no need (or use) for it on barcodes, and so we use the **aptoidexml** reference.

As a side note, the is no need for the XML header files, as only the tags are important. There for, all you have to do is translate the following structure:

```
aptoidexml://
<getapp>
     <name> … </name>
     <get> … </get>
</getapp>
<newserver>
     <server> … </server>
               ...
     <server> … </server>
</newserver>
```

An example might be:

Pretend you want to link the application “XPTO” witch is at “http://myrepo.my/xpto.apk” and want to distribute your repository – http://myrepo.my – with it.

Parse the following string to Qrcode:

aptoidexml://<getapp><name>XPTO</name><get>http://myrepo.my/xpto.apk</get></getapp><newserver> <server>http://myrepo.my</server> </newserver>

The resulting Qrcode would be:

NOTE: The QRcode functionality described above is supported by all Aptoide clients with version above 2.4.

[AD 5]Quick reference guide

[AD 1] Info.xml

	Mandatory?	Default Value	Information	Tag
Document tag	yes	- - -	The root tag for the info.xml file	<apklst>
Application enclosing tag	yes	- - -	The root tag for an application	<package>
Package	yes	- - -	The package that identify the application	<apkid>
Path	yes	- - -	The path to the *.apk on the server	<path>
Name	no	Equal to package	The human-readable application name	<name>
Version Code	no	0	Application version code. Used for upgrades.	<vercode>
Version Name	no	0.0	Application version name.	<ver>
Icon	no	Standard Android icon	The path to the icon file	<icon>
Main Category	no	Others	The main category of the application	<catg>
Secondary Category	no	Others	Se second category of the application	<catg2>
Date	no	01-01-00	The date the application entered the repository	<date>
Rating	no	3	The application rating (1 to 5 stars)	<rat>
Downloads	no	0	The number of downloads this apk file has on the server	<dwn>
Md5 hash	no	null	The md5hash of the apk file	<md5h>

[AD 2] Extras.xml

	Mandatory?	Default Value	Information	Tag
Document tag	Yes	- - -	The root tag for the extras.xml file	<extras>
Application enclosing tag	Yes	- - -	The root tag for an application	<pkg>
Package	Yes	- - -	The package that identify the application	<apkid>
Application Description	No	null	The description of the application	<cmt>

[AD 3] Supported categories

First Level Categories

- Games
- Applications
- Others

Note that category must be a string match to the ones above. If you failed to give a first lever category (<catg> tag on info.xml), or give one that don't compare to the ones above, the application will be inserted on the “Others” category.

Second Level Categories

The second level categories, as the first ones, must be a string match to the ones show below (they will be in the <catg2> tag on info.xml file). Is they do belong to a first level category but fail to match one of the one above, they will be inserted on the “Others” category inside the first level one:

Games:

Arcade & Action

Brain & Puzzle

Cards & Casino

Casual

Applications:

Comics
Communication
Entertainment
Finance
Health
Lifestyle
Multimedia
News & Weather
Productivity
Reference
Shopping
Social
Sports
Themes
Tools
Travel
Demo
Software Libraries