

PAW Functions

Für diese Seite existiert keine Übersetzung!

This page describes functions and object special to PAW and is meant for developers. Most of the examples can directly be tested.

Basics

With PAW it is possible to write dynamic web pages which can use the functionality of your phone.

Have a look at the folder /sdcard/paw/html/app on your SD card.

Some of the HTML files contain BeanShell code (enclosed in <bsh></bsh> tags).

Note: The directory /sdcard/paw/html/app will be deleted with new versions of PAW, so please put your own pages in a new directory below /sdcard/paw/html.

Here is an example how to retrieve information of the phone:

```
String[] opts = { "BOARD", "BRAND", "DISPLAY", "MODEL", "PRODUCT",
                 "TIME", "TYPE", "USER", "VERSION.INCREMENTAL",
                 "VERSION.RELEASE", "VERSION.SDK"};

for(o : opts) {
    print(o + ":" + eval("android.os.Build." + o));
}
```

Result:

Execute Script

Using the above example a simple xhtml web page could look like this:

```
<html>
<body>
<bsh>
String[] opts = { "BOARD", "BRAND", "DISPLAY", "MODEL", "PRODUCT",
                 "TIME", "TYPE", "USER", "VERSION.INCREMENTAL",
                 "VERSION.RELEASE", "VERSION.SDK"};

for(o : opts) {
    print("<b>" + o + " :</b>" + eval("android.os.Build." + o) + "<br>");
}
</bsh>

</body>
</html>
```

To retrieve the PAW Android Service object you can use the following code:

```
service = server.props.get("serviceContext");
```

To get a request parameter the following code can be used:

```
parameter = parameters.get("parameterName");
```

Get and Post parameters are combined and multiple variables are stored into an ArrayList.

Simplified BeanShell Tags

BeanShell pages can now include PHP like tags (<?bsh ... ?>).

A short print tag (<?=>) is also available.

Here is an example:

```
<html>
<body>
<?bsh
```

```

import de.fun2code.android.pawserver.AndroidInterface;
import android.content.Context;
import android.hardware.Sensor;

?>
<?bsh
    sensorListener = AndroidInterface.getSensorListener();
    proximity = sensorListener.getProximity();

    if(proximity > 0) {
?>
<h3> Hi ... :) </h3>
<?bsh
    }
?>
<pre>Proximity: <?=proximity ?></pre>
</body>
</html>

```

Deploying Web Pages

Web pages can be copied to the paw html folder of the SD card (/sdcard/paw/html).

Server Startup

Server startup scripts are located in the /sdcard/paw/etc/init directory. There are two runlevels 0 and 1. Runlevel 0 is for pre startup/shutdown scripts and 1 for post startup/shutdown sripts. Scripts with the prefix S_ will be processed on startup. Scripts prefixed by K_ will be processed on shutdown.

Note: The server variable is not available for pre-startup scripts.

Application Configuration

PAW Server supports multiple web applications.

Web applications can be configures in the /sdcar/paw/webconf/apps directory inside a conf file.

As example have a look at the standard PAW configuration file.

The conf file have the follwing structure:

```

name=<Application name>>
description=<Application description>
icon=<Path to application ion (should be 48x48px)>
href=<Application path>

```

Example:

```

name=Test Application
description=Test application
icon=app/images/android.png
href=app/

```

If more than one application is configured a menu will be displayed when http:<ip-number>:<port> is called:



Enhanced BeanShell Debug Output

If the BeanShell handler in paw/conf/handler.xml contains the property debug with the value true the error output that is included inside the page source is enhanced. This now includes the generated BeanShell source together with line numbers and the erroneous line market with an asterisk (*).

Here is an exmample:

```

<!--
Generated BeanShell Code:

1  $$.$$.print("<html>\n<body>\n");
2  // Imports

```

```

3  import de.fun2code.android.pawserver.AndroidInterface;
4  import android.content.Context;
5  import android.hardware.Sensor;
6  sensorListener = AndroidInterface.getSensorListener();
* 7  proximity = sensorListene.getProximity();

-->

```

BeanShell Handlers

The `org.paw.handler.CallBshHandler` makes it possible to write handlers in BeanShell. This eliminates the need to comile code. It is recommended to use this only for testing purposes, because BeanShell code might have some performance issues. The handler can be inserted into the `paw/conf/handler.xml` file.

Here is an example:

```

<handler status="active">
  <name>Call BeanShell Handler Wrapper</name>
  <description>Call BeanShell Handler</description>
  <removable>true</removable>
  <id>callBshHandler</id>
  <files/>
  <params>
    <param name="callBshHandler.class" value="org.paw.handler.CallBshHandler" />
    <param name="callBshHandler.script" value="[PAW_HOME]/bshHandlerFilter/testHandler.bsh" />
  </params>
</handler>

```

The handler parameter `script` defines the BeanShell script to call. The BeanShell script contains the same method definitions as defined by the Brazil Handler Interface.

For more information, have a look at the [Brazil API](#).

Below is an empty script which shows the methods and the variales available for each method.

```

/*
  Variables: server, prefix, handler
*/
init() {

}

/*
  Variables: server, prefix, handler, request
*/
respond() {

}

```

To all methods the variable `handler` is passed, which holds a reference to the calling *Handler*. To store objects inside the handler you can use the method `handler.save(String key, Object object)`. To load an object form the Handler use `handler.load(String key)`.

BeanShell Filters

Similar to the BeanShell Handler there is also the BeanShell Filter class `org.paw.filter.CallBshFilter`. This filter works the same way as the above described BeanShell Handler and is defined inside the `paw/conf/filter.xml` file.

Here is a sample filter.xml entry:

```

<filter status="active">
  <name>Call Bsh Filter</name>
  <description>Sample CallBshFilter</description>
  <removable>true</removable>
  <id>callbshFilter</id>
  <files />
  <params>
    <param name="callbshFilter.class" value="org.paw.filter.CallBshFilter" />
    <param name="callbshFilter.script" value="[PAW_HOME]/bshHandlerFilter/statusPageFilter.bsh" />
  </params>
</filter>

```

The parameter `script` defines the BeanShell script to call which includes methods implemented by the Filter interface provided by the [Brazil API](#).

Here is an empty script containing the available methods and provided variables.

```

/*
  Variables: server, prefix, filter
*/

```

```

init() {
}

/*
  Variables: server, prefix, request, filter
*/
respond() {
}

/*
  Variables: server, prefix, headers, filter
*/
shouldFilter() {
}

/*
  Variables: server, prefix, request, headers, content, filter
*/
filter() {
}

```

The variable `filter` is passed to all methods. To persist objects inside the filter, use the `filter.save(String key, Object object)` and `filter.load(String key)` methods.

File Not Found Handler

The new `org.paw.handler.FileNotFoundHandler` is called if a document is not handled by any other handler and is located at the end of the handler chain.

The default configuration returns a default error message. To set a custom message the parameter `message` can be used.

Here is an example:

```
<param name="notFound.message" value="Sorry, page not Found." />
```

To configure a custom web page the parameters `file` and `mime` can be defined.

Example:

```
<param name="notFound.file" value="[PAW_HOME]/html/pages_err/not_found.html" />
<param name="notFound.mime" value="text/html" />
```

The following placeholders can be put into a custom HTML page:

```

@status    HTTP status code

@url       Requested URL

@server    Server name and version

```

Plug-ins

The webapp supports plug-ins. Plugins are located in the `/sdcard/paw/html/app/plugins` directory. Each plug-in contains a `plugin.conf` file. The structure of the file is the same as for webapps.

If plug-ins are detected they are available in a separate **Plug-ins** menu entry.

A sample plug-in can be downloaded [here](#).

Autostart

The Server supports Autostart of scripts if enabled in the server settings.

Scripts are executed in alphabetic order. They reside inside the `/sdcard/paw/autostart` directory and have the file extension `bsh`.

A test script is available [here](#). After renaming the script to have a `bsh` extension and placing it inside the autostart folder and enabling the server setting the script will be automatically started at next server startup.

The script starts a background thread that displays the server uptime in the notification bar of the device.

Example script:

```

import de.fun2code.android.pawserver.PawServerService;

/*
 * Test autostart script that displays the server's uptime as notification
 */

r = new Runnable() { public void run() {
    start = System.currentTimeMillis();
    notificationId = new Random().nextInt(1000);

    while(PawServerService.isRunning()) {
        uptime = (System.currentTimeMillis() - start) / 1000;

showNotification(com.android.internal.R.drawable.ic_menu_emoticons, "PAW Server
Info", "PAW Server", "PAW Server Uptime: " + uptime + " sec", notificationId,
true, "#FF0000", false, false);
        Thread.sleep(1000);
    }

    cancelNotification(notificationId);
}};
t = new Thread(r);
t.start();

```

Bootup Preference Injection

To make it easier to change app preferences without starting the app a mechanism was implemented to inject preference on bootup of the device.

Note: This is only done at bootup of the device and not on each start of the app.

Preferences that should be set can be specified inside the file PAW_HOME/etc/preferences.

The file follows the Java properties file definition. The types String, Integer, Float and Boolean are auto detected.

This mechanism can also be used within own web apps. The preferences are located inside the shared preferences within the PAW application.

The preference is named PawServer.

Here is a sample preferences file with the available PAW preferences:

```

#-----
# Allows to change PAW preferences upon boot.
#-----
#AutoStart=true
#HideNotificationIcon=true
#ShowUrlInNotification=true
#ExecAutostartScripts=true
#UseWakeLock=true
#FreezeContent=true

```

Dynamic DEX Class Loading

Note: Due to a [bug](#) in Honeycomb, dynamic class loading might not work.

Class Loading on Startup

Classes in DEX format included in JAR and APK files can be dynamically loaded on startup.

All JAR and APK files present in the folder paw/webconf/dex will be added to a new classloader called dexClassLoader.

The dexClassLoader can be retrieved by calling server.props.get("serviceContext").getDexClassLoader().

To use the class loaded by this classloader simply use the BeanShell command useDexClasses().

Dynamic Handler

There is a new Brazil Handler class called org.paw.handler.AndroidDynamicHandler that allows the wrapping of Handlers that are only available in external DEX format.

The Handler has two parameters:

- handlerJars: Colon separated list of JAR or APK files to load. This parameter is optional. If all necessary classes are available by the dexClassLoader (see "Dynamic DEX Class Loading"), this parameter is not necessary.
- handlerClass: The Handler class to use.

All other parameters defined in the Handler definition are passed to the Handler defined in the handlerClass parameter.

Dynamic Filter

Equivalent to the Dynamic Handler but for Filters. The class is called org.paw.filter.AndroidDynamicFilter. The Filter has two parameters:

- `filterJars`: Colon separated list of JAR or APK files to load. This parameter is optional. If all necessary classes are available by the `dexClassLoader` (see "Dynamic DEX Class Loading"), this parameter is not necessary.
- `filterClass`: The Filter class to use.

All other parameters defined in the Filter definition are passed to the Filter defined in the `filterClass` parameter.

Native Mapper

The Native Mapper allows to map URLs to native DEXed classes allowing for a much better performance than interpreted BeanShell code. To use the Native Mapper, create a DEXed jar file and put it inside the `/sdcard/paw/webconf/dex` directory.

DEXed classes can be created by using the `dx` command from the Android SDK:

```
./dx --dex --output=out_dex.jar --positions=lines in.jar
```

The mapping file has the following format and has to be placed in the `/sdcard/paw/webconf/mapping` directory:

```
URL=DEX class name
```

All files present in that directory will be loaded by the Native Mapper.

Here is an example file content from the Web Application:

```
/app/get_app_icon.html=de.fun2code.android.pawserver.external.webapp.graphics.FetchAppIcon
/app/graphics/contact_photo.html=de.fun2code.android.pawserver.external.webapp.graphics.FetchContactPhoto
/app/get_scaled_image.html=de.fun2code.android.pawserver.external.webapp.graphics.FetchScaledImage
/app/graphics/albumart.html=de.fun2code.android.pawserver.external.webapp.graphics.FetchAlbumArt
/app/graphics/system_load_image.html=de.fun2code.android.pawserver.external.webapp.graphics.DrawLoadImage
```

The class that is called must define a method with the name `response` and the following signature:

```
public void respond(Request request, Server server, Hashtable<String, Object> parameters)
```

All the variables available in BeanShell XHTML files are also passed to this method.

Scoping

The server noch supports two scopes. A Server and a Session scope.

Objects can be placed inside the scoped with the following commands:

```
serverPut(key, object);
serverGet(key);
```

```
sessionPut(key, object);
sessionGet(key);
```

BeanShell

BeanShell now has an additional variable `$$` which is an alias for the BeanShell interpreter.

This can be used to generate output without linebreak by using the `$$.print ()` command.

To get request parameters the parameter `Hashtable` is provided. The till now uses statement `request.getQueryData ()` is deprecated, because there were encoding problems with this function.

As an example: To get the parameter value of a parameter called `text` you would use: `parameters.get ("text")`;

General Functions

MD5

The command `md5(string)` accepts a `string` as argument and converts it into a MD5 hash.

```
$$ .print(md5("test string"));
```

Execute Script

Result:

Android Functions

In this version interacting with Android has been improved.

For some of these functions it is necessary that the PAW Server activity is active in the foreground. The activity is therefore automatically, if needed.

Below are examples which demonstrate these commands. To test them yourself you can change the code inside the text fields.

Scan Barcode

For this functionality to work, the Barcode Scanner application has to be installed.

If this is not the case, you can press the following button and the Android Market will be opened on your device: [Install Barcode Scanner](#)

```
res = scanBarcode();
```

Result:

Display Toast Message

As duration parameter "short" or "long" can be used.

```
makeToast("Test Toast message...", "short");
```

Display Alert Dialog

```
showAlert("Dialog Title", "The text message.");
```

Display Alert Dialog with Input

```
res=showInputDialog("Dialog Title", "Type in some text:");
```

Result:

Send an Email

```
sendMail("android@fun2code.de", "Test Mail from PAW Server", "Hello,\nThis is
```

Open an URI

```
openUri("market://search?q=pname:de.fun2code.android.pawserver");
```

Dialing a Number

Opens the dialer app, the user has to start the call manually.

The phone number is provided as parameter.

As a demo the below script calls the USSD code *100#. This code might not be available on all carriers.

```
dial("*100#");
```

Calling a Number

Calls the phone app on the phone and dials the number right away. The phone number is provided as parameter.

As a demo the below script calls the USSD code *100#. This code might not be available on all carriers.

```
call("*100#");
```

Ending a Phone Call

The BeanShell command `endCall()` ends a phone call.

Note: This uses unofficial API calls!

```
endCall();
```

Accelerometer Support

Accelerometer support has been added in this release.

The sensor instance can be obtained by calling `de.fun2code.android.pawserver.AndroidInterface.getSensorListener()`.

The SensorListener has the following methods:

- getX()
- getY()
- getZ()
- getForce()

Force calculation is experimental. The force is calculated like this:

```
Math.abs(dataX + dataY + dataZ - last_x - last_y - last_z) / diffTime * 10000;
```

There is also a web service (/app/ws/sensor_data.xhtml) which return a JSON array.

JSON output:

```
{"x": -0.0139617919921875, "y": -0.38800048828125, "z": 9.494155883789062, "force": 7.19451904296875}
```

Example

The below example shows the orientation of your phone. Turn your phone and the image should turn as well.



Media Player support

It is now possible to get information from and control the Android Media Player. This is using undocumented classes, so support might break in future Android releases.

To get the Media Player object the following code can be used:

```
import de.fun2code.android.pawserver.AndroidInterface;  
AndroidInterface.getMediaPlayerService().getPlayBackService();
```

The Media Player object supports the following methods:

```
void openfile(String path);  
void openfileAsync(String path);  
void open(in int [] list, int position);  
int getQueuePosition();  
boolean isPlaying();  
void stop();  
void pause();  
void play();  
void prev();  
void next();  
long duration();  
long position();  
long seek(long pos);  
String getTrackName();  
String getAlbumName();  
int getAlbumId();  
String getArtistName();  
int getArtistId();  
void enqueue(in int [] list, int action);
```



```

int [] getQueue();
void moveQueueItem(int from, int to);
void setQueuePosition(int index);
String getPath();
int getAudioId();
void setShuffleMode(int shufflemode);
int getShuffleMode();
int removeTracks(int first, int last);
int removeTrack(int id);
void setRepeatMode(int repeatmode);
int getRepeatMode();
int getMediaMountedCount();

```

Speech To Text

Checking for available Intents and speech to text support have been implemented.
The commands are:

```

boolean isIntentAvailable(String intent)
String speechToText(String title)

```

Below are examples which demonstrate these commands. To test them yourself you can change the code inside the text fields.

```

import android.speech.RecognizerIntent;

if(isIntentAvailable(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)) {
    res = speechToText("Text to speech");
}
else {
    res = "SpeechToText not supported!";
}
$$$.print(res);

```

Result:

Execute Script

Save Bookmark

The saveBookmark(String title, String url) command saves a bookmark in the phones browser.

Below is an example that demonstrates this command. To test them yourself you can change the code inside the text fields.

```

saveBookmark("http://www.fun2code.de", "Fun2Code");

```

Execute Script

XML-RCP support

The server now supports XML-RCP from <http://code.google.com/p/android-xmlrpc>.
For an example have a look at the [XML-RCP code snippet](#).

GPS

Functions for activating GPS have been added. The GPS has to be enabled manually by the user on the device before this can work. When the function activateGPS(true) is used the GPS icon should appear in the statusbar.

The function activateGPS(false) will stop the device from receiving GPS signals. The GPS icon should disappear in the statusbar.
To get the last GPS position the command getGpsLocation() which returns an Android Location object can be used.

Below code will start the GPS and stop it again after 5 seconds.

```

activateGps(true);
Thread.sleep(5000);
activateGps(false);

```

Execute Script

WiFi

To switch WiFi on and off use the command enableWifi(boolean) can be used.
This of course only makes sense if the PAW server is not running on WiFi.

Below code will stop WiFi and restart it after 2 seconds.

```
enableWifi(false);
Thread.sleep(2000);
enableWifi(true);
```

Execute Script

Orientation Sensor

In addition to GPS functions the compass orientation can be queried. The following script retrieves the current bearing every two seconds.

```
import de.fun2code.android.pawserver.AndroidInterface;

sensorListener = AndroidInterface.getSensorListener();
bearing = sensorListener.getOrientBearing();
$$.$$.print(bearing);
```

Result:

196.0095

Bearing of device:



The sensor listener provided the following orientation methods:

- `getOrientBearing()` - Returns the bearing
- `getOrientAccuracy()` - Gets the accuracy (0=unreliable, 1=low, 2=medium, 3=high)
- `getLastOrientTimestamp()` - Timestamp of last update in millis

Proximity Sensor

To get the value of the proximity sensor the method `getProximity()` is provided by the sensor listener.

```
import de.fun2code.android.pawserver.AndroidInterface;

sensorListener = AndroidInterface.getSensorListener();
$$.$$.print( sensorListener.getProximity());
```

Result:

Execute Script

Light Sensor

To get the brightness in Lux of the light sensor the method `getBrightness()` is provided by the sensor listener.

```
import de.fun2code.android.pawserver.AndroidInterface;

sensorListener = AndroidInterface.getSensorListener();
$$.$$.print( sensorListener.getBrightness());
```

Result:

Execute Script

Pressure Sensor

To get the pressure in hPa (millibar) of the pressure sensor (barometer) the method `getPressure()` is provided by the sensor listener.

```
import de.fun2code.android.pawserver.AndroidInterface;
sensorListener = AndroidInterface.getSensorListener();
$.print(sensorListener.getPressure());
```

Result:

Execute Script

Gyroscope Sensor

Values from the Gyroscope sensor are provided by the following methods of the sensor listener:

- `getGyroX()`: Angular speed around the x-axis (radians/second).
- `getGyroY()`: Angular speed around the y-axis (radians/second).
- `getGyroZ()`: Angular speed around the z-axis (radians/second).

```
import de.fun2code.android.pawserver.AndroidInterface;
sensorListener = AndroidInterface.getSensorListener();
print("Gyro X: " + sensorListener.getGyroX());
```

Result:

Execute Script

Text To Speech

Text To Speech is now supported by the `speak(text, locale)` method.

Note: Text To Speech is only supported since Android 1.6.

```
speak("This is a test message", new Locale("en"));
```

Execute Script

Relative Humidity Sensor

To get the relative humidity of the humidity sensor (in %) the method `getRelativeHumidity()` is provided by the sensor listener.

```
import de.fun2code.android.pawserver.AndroidInterface;
sensorListener = AndroidInterface.getSensorListener();
$.print(sensorListener.getRelativeHumidity());
```

Result:

Execute Script

Ambient Temperature Sensor

To get the ambient temperature of the temperature sensor (in °C) the method `getAmbientTemperature()` is provided by the sensor listener.

```
import de.fun2code.android.pawserver.AndroidInterface;
sensorListener = AndroidInterface.getSensorListener();
$.print(sensorListener.getAmbientTemperature());
```

Result:

Execute Script

Camera Preview

Note: Works only on Froyo (Android 2.2) and up!

A simple script than demonstrates the use of the camera preview as public webcam can be found [here](#). Just download it to your /sdcard/paw/html directory and change the file extension to xhtml.

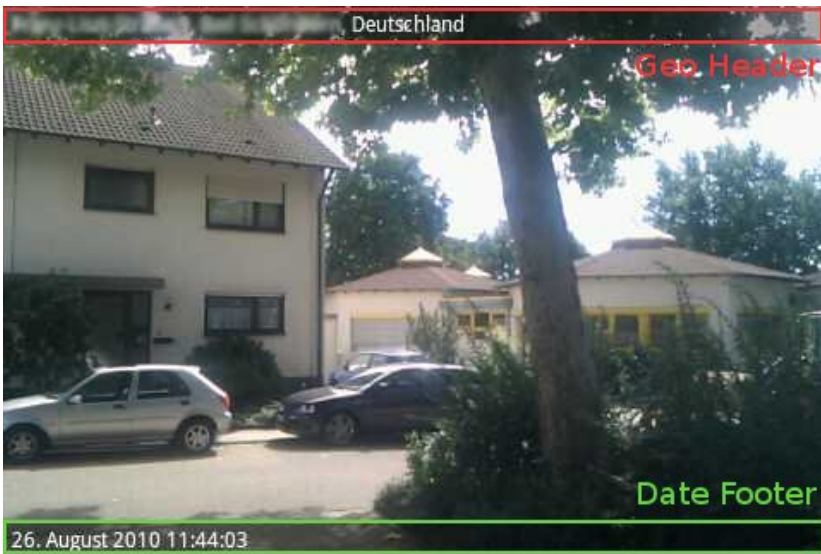
The camera is not a shared resource, so the camera has to be stopped (closed) before it can be reused by another application!

The CameraPreview class in the `de.fun2code.android.pawserver.media` package provides the following methods:

<code>CameraPreview getInstance()</code>	Return the CameraPreview instance. Because the camera is not a shared resource this is a singleton.
<code>setFile(File file)</code>	Sets the output file.
<code>File getFile()</code>	Returns the output file.
<code>setBackupDirectory(File dir)</code>	Sets the backup directory which stores copies of the images. The directory must exist. File format is: <code>yyyyMMdHHmss.<ext></code>
<code>File getBackupDirectory()</code>	Returns the backup directory.
<code>setQuality(int quality)</code>	Sets the image quality in %. Has to be between 0 and 100. Defaults to 100.
<code>int getQuality()</code>	Gets the image quality in %.
<code>setRotation(int deg)</code>	Sets the image rotation in degrees. 0 is the default rotation (device in landscape, camera to the left).
<code>int getRotation()</code>	Gets the image rotation in degrees.
<code>setSize(int width, int height)</code>	Sets the preview size. Has to be a valid size! Valid sizes can be retrieved by the <code>getSupportedSizes()</code> method.
<code>List getSupportedSizes()</code>	Returns a list of supported sizes.
<code>setFormat(Bitmap.CompressFormat format)</code>	Sets the image output format. Defaults to JPEG.
<code>Bitmap.CompressFormat getFormat()</code>	Returns the image output format.
<code>setInterval(Integer interval)</code>	Defines how often images are written. Defaults to 0.
<code>setPrintGeoLocation(boolean enable, String foregroundRGB, String backgroundRGB)</code>	Specifies if and in which color the geo location header is printed.
<code>setCameraId(id)</code>	Sets the camera ID. The default is 1. This might be useful if the front-facing-camera should be used instead of the default.
<code>int getCameraId()</code>	Gets the camera ID. The default is 1.
<code>Camera getCamera()</code>	Returns the camera object, or null if preview has not been started.
<code>boolean start()</code>	Acquires the camera and starts the preview. Returns true on success otherwise false.
<code>stop()</code>	Stops the preview and releases the camera.
<code>boolean isCameraIdSupported()</code>	Checks if camera IDs (since Gingerbread) are supported. Returns true on success, otherwise false.
<code>int getBackCameraId()</code>	Returns the ID of the back facing camera. If no camera is found -1 is returned.
<code>int getFrontCameraId()</code>	Returns the ID of the front facing camera. If no camera is found -1 is returned.
<code>void setSurfaceEnabled(boolean state)</code>	The surface view can be enabled or disabled. Without surface, camera preview might not work on some devices! It's enabled by default.
<code>void setSurfaceSize(int width, int height)</code>	Sets the size of the surface view. Default is width=-1 and height=-1 which results in a full screen view.

The preview supports header and footer information. If specified the header contains geo information provided by the Google Geocoder. The footer contains the date if specified.

Below is a sample image:



The following script will create an image in the /sdcard directory with the name webcam_test.jpg with the size of 320x240 pixels, a geo header, date footer and the image quality of 50%.

It starts the camera preview, waits four seconds and stops the preview.

You can use the buttons below the script to view and delete the image.

```
import de.fun2code.android.pawserver.media.*;

cp = CameraPreview.getInstance();
cp.setSize(320, 240);
cp.setFile(new File("/sdcard/webcam_test.jpg"));
cp.setQuality(50);
cp.setPrintDate(true, "#ffffff", "#55000000");
cp.setPrintGeoLocation(true, "#ffffff", "#55000000");
cp.start();
Thread.sleep(4000);
cp.stop();
```

Execute Script

Show Image

Delete Image

Notifications

Notifications are supported by three methods.

A [Notification Builder](#) is available in the *Code Snippets* section.

showNotification(ticker, header, text)

This is a simple function and generates a simple text message. It's equivalent to calling showNotification(null, ticker, header, text, null, true, null, true). It returns the notification id.

showNotification(icon, ticker, header, text, id, light, color, sound, vibrate)

Generates a notification with the following parameters:

- icon: Icon resource id. If null de.fun2code.android.pawserver.R.drawable.notification is used.
- ticker: Ticker text. If null no notification will be displayed in the status bar.
- header: Header text. If null no notification will be displayed in the status bar.
- text: Text displayed beneath the header. If null no notification will be displayed in the status bar.
- id: Notification id. If null a random id will be generated.
- light: true if LED light should be used, otherwise false.
- color: LED RGB color (#rrggbb). If null standard color will be used.
- sound: true if sound should be used, otherwise false.
- vibrate: true if vibration should be used, otherwise false.

It returns the notification id.

cancelNotification(id)

Cancels the notification with the given id.

The below script will display two notifications, one with the simple and one with the extended method. After 15 seconds the notifications will be cleared.

```
id1 = showNotification("Test Notification 1", "Notification Header",
"Notification Text");

id2 = showNotification(com.android.internal.R.drawable.ic_menu_notifications,
"Test Notification 2", "Notification Header", "Notification Text", null, true,
"#0000FF", true, true);

Thread.sleep(15000);

cancelNotification(id1);
cancelNotification(id2);
```

Execute Script

Vibration

Vibration is supported by the simple method `vibrate(millis)` which causes the device to vibrate for the given number of milliseconds. The script below vibrates for 3 seconds.

```
vibrate(3000);
```

Execute Script

Image Filters

Image filters for grayscale and sepia are available.

The two methods `Graphics.filterBitmapSepia(String)` and `Graphics.filterBitmapGrayScale(String)` in the package `de.fun2code.android.pawserver.util.Graphics` take the filename (as `String`) and return a `android.graphics.Bitmap` as result.

Clipboard

With the methods `String readFromClipboard()` and `writeToClipboard(String)` it is now possible to read from and write to the the clipboard of the device.

The example will write a text to the clipboard of the device and read it afterwards.

```
writeToClipboard("Clipboard Text");
$.print(readFromClipboard());
```

Result:

Execute Script

Send SMS Message

The following command sends a SMS message and stores it in the outbox. The return value indicates if the message could be sent.

```
boolean sendSMS(String number, String message)
```

To send a SMS message that is not stored in the outbox the following command with `silent` set to `true` can be used:

```
boolean sendSMS(String number, String message, boolean silent)
```

SMS Actions

SMS Actions are scripts that are executed when a SMS message with the correct **Name** and **PIN** combination arrives. Actions can be registered with `SmsListener` available in the `de.fun2code.android.pawserver.listener` package.

The SMS has to have the following format to be processed by an action:
ActionName:PIN:[Additional Arguments]

The scripts have access to the following additional variables:

- `smsArgs`: Additional arguments that can be added after the last colon.
- `smsNumber`: The phone number of the SMS sender.

The `SmsListener` provides the following Action methods:

- `boolean registerAction(String name, String pin, String code)`: Registers a new Action. Returns `true` on success, otherwise `false`.
- `boolean updateAction(String name, String pin, String code)`: Updates an existing Action. Returns `true` on success,

- otherwise false.
- boolean `removeAction(String name)`: Updates an existing Action. Returns true on success, otherwise false.
- `TreeMap getActions()`: Returns all Actions as `TreeMap`.
- `clearActions()`: Clears all Actions.
- `testMessage(String phoneNumber, String smsBody)`: Tests an Action.

The example registers a SMS Action which displays the additional arguments of a SMS message as Toast message if the message has the following format:
reply:123:Test arguments...

The last command tests the message.

```
import de.fun2code.android.pawserver.listener.*;

SmsListener.registerAction("test", "123", "makeToast(\"Number: \" + smsNumber +
\" Args: \" + smsArgs, \"short\");");
SmsListener.testMessage("5463", "test:123:Test arguments...");
```

Execute Script

SMS Scripts

SMS Scripts are scripts that are executed in alphabetic order each time a SMS message arrives. Scripts can be registered with `SmsListener` available in the `de.fun2code.android.pawserver.listener` package.

The scripts have access to the following additional variables:

- `smsMessage`: SMS body.
- `smsNumber`: The phone number of the SMS sender.

The `SmsListener` provides the following Script methods:

- boolean `registerScript(String name, String code)`: Registers a new Script. Returns true on success, otherwise false.
- boolean `updateScript(String name, String code)`: Updates an existing Script. Returns true on success, otherwise false.
- boolean `removeScript(String name)`: Updates an existing Script. Returns true on success, otherwise false.
- `TreeMap getScripts()`: Returns all Scripts as `TreeMap`.
- `clearScripts()`: Clears all Scripts.
- `testMessage(String phoneNumber, String smsBody)`: Tests the Scripts.

The example registers a SMS Scripts which displays the number and message as Toast message on the display of the device. The last line is used to test the Script.

```
import de.fun2code.android.pawserver.listener.*;

SmsListener.registerScript("test", "makeToast(\"Number: \" + smsNumber + \"
Message: \" + smsMessage, \"short\");");
SmsListener.testMessage("5463", "SMS message...");
```

Execute Script

Sharing

The shared Intents are stored in a `HashMap<Long, Intent>` in the server scope. The key of the map is a timestamp which can be used to display the date when the content has been shared. To receive the map use the following code:

```
server.props.get("de.fun2code.android.pawserver.key.share")
```

Broadcast Listeners

The following methods are available in the `de.fun2code.android.pawserver.util.IntentUtil` class to handle broadcast events:

- static `Intent getIntentResultFromBroadcast(context, intentAction)` - Waits for a single Intent to arrive and provides this as return value. The integer result code is available as extra data in the resulting intent ("result").
- static `Intent getIntentResultFromBroadcast(context, intentAction, millis)` - Waits for a single Intent for the specified amount of time and provides this as return value. The integer result code is available as extra data in the resulting intent ("result").
- static `List getIntentResultsFromBroadcast(context, intentAction, millis)` - Collects Intent for the specified amount of milli seconds and returns them as List. The integer result code is available as extra data in the resulting intents ("result").

To create `BroadcastListener` which executes a BeanShell script in the `onReceive()` method when an Intent is received the method `buildReceiver` from the `de.fun2code.android.pawserver.util.BroadcastReceiverUtil` class can be used. The intent and context objects are available from within the script.

- static `BroadcastListener buildReceiver(script)`;

The example shows how to use the `IntentUtil.getIntentResultFromBroadcast()` to receive the battery status in a single Intent

an prints the result. It also registers two BroadCastListeners by using `BroadcastReceiverUtil.buildReceiver()` for five seconds that will display and remove notifications if power is plugged or unplugged.

```
import de.fun2code.android.pawserver.util.*;
import android.content.Intent;
import android.content.IntentFilter;

service = server.props.get("serviceContext");

// Get single Intent
intent = IntentUtil.getIntentResultFromBroadcast(service,
Intent.ACTION_BATTERY_CHANGED);
level = intent.getIntExtra("level", 0);
```

Result:

Execute Script

NFC

Reading of NFC tags is supported by the `AndroidInterface` class method `Intent getNfcIntent(int timeoutSec)`. The method expects a timeout parameter in seconds.

Here is an example, that shows how to retrieve the Tag from the resulting Intent:

```
import de.fun2code.android.pawserver.AndroidInterface;
import android.nfc.NfcAdapter;

intent = AndroidInterface.getNfcIntent(5); // 5 secs timeout
if(intent != null) {
    tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
    print(tag);
}
else {
    print("No Tag found!");
}
```

Result:

Execute Script

NDEF Records

Note:

The purpose of the NDEF record classes is to simplify reading and writing of NDEF messages.
A sample NFC reader/writer application is available as [plugin](#).

The package `de.fun2code.android.pawserver.media.nfc.ndef` contain the following classes to support NDEF records:

<code>TextRecord</code>	NDEF Text record.
<code>URIRecord</code>	NDEF URI records.
<code>SmartPosterRecord</code>	NDEF Smart poster records.
<code>MimeMediaRecord</code>	NDEF Mime Media poster records.

All record types contain a `writeTag(Tag)` method which returns one of the following codes defined in the class `de.fun2code.android.pawserver.media.nfc.Constants`:

<code>SUCCESS</code>	Write was successful.
<code>ERROR</code>	An Error occurred.
<code>MESSAGE_TOO_LARGE</code>	Message was too large, which means longer than the maximum NDEF message length.

TextRecord

Text NFEF record which takes text and language code as argument. The following methods are defined:

The constructor. Arguments are language code (e.g. "en") and

<code>TextRecord(String languageCode, String text)</code>	text.
<code>String getText()</code>	Returns the text.
<code>String getLanguageCode()</code>	Returns the language code.
<code>static TextRecord parse(NdefRecord record)</code>	Parses a NdefRecord and returns a TextRecord instance.
<code>static boolean isText(NdefRecord record)</code>	Checks if the given NdefRecord is a NDEF Text record.
<code>static NdefRecord createRecord(String text, String languageCode)</code>	Creates a new Text NdefRecord
<code>int writeTag(Tag tag)</code>	Writes the tag and returns the result code (see Constants mentioned above.)

UriRecord

URI NDEF record which takes the URI as argument. The following methods are defined:

<code>UriRecord(Uri uri)</code>	The constructor. Arguments an URI.
<code>Uri getUri()</code>	Returns the URI.
<code>static UriRecord parse(NdefRecord record)</code>	Parses a NdefRecord and returns a UriRecord instance.
<code>static boolean isUri(NdefRecord record)</code>	Checks if the given NdefRecord is a NDEF URI record.
<code>static NdefRecord createRecord(Uri uri)</code>	Creates a new URI NdefRecord
<code>int writeTag(Tag tag)</code>	Writes the tag and returns the result code (see Constants mentioned above.)

SmartPosterRecord

Smart Poster NDEF record which takes URI, text and language code as argument. The following methods are defined:

<code>SmartPosterRecord(Uri uri, String languageCode, String text)</code>	The constructor. Arguments are URI, language code (e.g. "en") and text.
<code>Uri getUri()</code>	Returns the URI.
<code>String getText()</code>	Returns the text.
<code>String getLanguageCode()</code>	Returns the language code.
<code>static SmartPosterRecord parse(NdefRecord record)</code>	Parses a NdefRecord and returns a SmartPosterRecord instance.
<code>static boolean isText(NdefRecord record)</code>	Checks if the given NdefRecord is a NDEF Smart Poster record.
<code>static NdefRecord createRecord(Uri uri, String languageCode, String text)</code>	Creates a new Smart Poster NdefRecord
<code>int writeTag(Tag tag)</code>	Writes the tag and returns the result code (see Constants mentioned above.)

MimeMediaRecord

Mime Media NDEF record which takes a mimetype and byte array data as argument. The following methods are defined:

<code>MimeMediaRecord(String mimeType, byte[] data)</code>	The constructor. Arguments are mimetype and byte array.
<code>String getMimeType()</code>	Returns the mimetype.
<code>byte[] getData()</code>	Returns the binary data.
<code>static MimeMediaRecord parse(NdefRecord record)</code>	Parses a NdefRecord and returns a MimeMediaRecord instance.
<code>static boolean isText(NdefRecord record)</code>	Checks if the given NdefRecord is a NDEF Mime Media record.
<code>static NdefRecord createRecord(String mimeType, byte[] data)</code>	Creates a new Mime Media NdefRecord
<code>int writeTag(Tag tag)</code>	Writes the tag and returns the result code (see Constants mentioned above.)

WebSockets

PAW supports WebSockets with a WebSocket handler that can be inserted into the handler.xml file.

The handler has two parameters, config and basedir. The config parameter defines where the WebSocket configuration is located, the basedir parameter defines the root directory for BeanShell scripts.

Here is a sample handler configuration, which should be located before the authentication handler:

```
<handler status="active">
  <name>WebSocket Handler</name>
  <description>WebSocket Handler</description>
  <removable>true</removable>
  <id>websocket</id>
  <files/>
  <params>
    <param name="websocket.class" value="org.paw.handler.android.websocket.WebSocketHandler" />
    <param name="websocket.basedir" value="[PAW_HOME]" />
    <param name="websocket.config" value="[PAW_HOME]/webconf/websocket.conf" />
  </params>
```

```
</handler>
```

WebSocket Configuration

The WebSocket configuration file is a simple textfile with the following format:

```
<WebSocket protocol>:<BeanShell file (extension .bsh) or native class>
```

The first parameter is the WebSocket protocol which defined by the `sec-websocket-protocol` header field. Regular Expression are supported as protocol name.

The second is a BeanShell script with the extension `.bsh` or a native class.

Sample configuration file:

```
# Format:
# Protocol:Bsh script (.bsh extension) or class name
#
chat.*:websocketTest.WebSocketTest
echo:html/websocket_echo.bsh
```

BeanShell Scripts

The easiest way to use WebSockets are BeanShell scripts. These are quite slow, so in a production environment you should consider using native classes.

The following variables are passed: `action`, `protocol`, `socket`, `sockets`, `message`.

The `action` variable can be "connect", "disconnect", "text" or "binary". The value "connect" is set if a client connects to a socket and "disconnect" if a client disconnects. In addition the used protocol name and the `Socket` are passed. The `sockets` and `message` variables are not present in these cases.

If a text or binary message is received, the `message` and `sockets` variables are passed. The variable `message` contains the text or binary message as *String*. All sockets belonging to the protocol are stored inside the `sockets` variable which is a *List of Sockets*.

To send a message to a socket, the following call can be used:

```
de.fun2code.android.pawserver.websocket.WebSocketMessage.sendMessage(message, socket)
```

Here is a sample script, which sends the received message to all connected clients:

```
if(action.equals("text")) {
    for(sock : sockets) {
        try {
            de.fun2code.android.pawserver.websocket.WebSocketMessage.sendMessage(message, sock);
        }
        catch(e) {}
    }
}
```

Web Sockets: Get all Sockets for Protocol

The command `wsSockets(protocol)` return a list of all sockets belonging to the specified protocol (e.g. "chat").

Web Sockets: Send Message to All Protocol Sockets

The BeanShell command `wsSendAll(message, protocol)` sends a message to all sockets belonging to the specified protocol.

Native Classes

Using native classes significantly increases performance, but they are not as easy to use in comparison to BeanShell scripts.

To get started, put the following JAR file to the classpath of your project: [pawWebSocket.jar](#)

After the code is compiled, you have to convert the classes into DEX format and put them into the `PAW_HOME/webconf/dex` directory. A detailed description can be found inside the following blog post: [PAW - Dynamic DEX Class Loading](#)

Here is a sample Java classfile that does exactly the same as the BeanShell sample above:

```
import java.io.IOException;
import java.net.Socket;
import java.util.List;

import de.fun2code.android.pawserver.websocket.WebSocketListener;
import de.fun2code.android.pawserver.websocket.WebSocketMessage;

public class WebSocketTest implements WebSocketListener{

    @Override
    public void onConnect(Socket socket, String protocol) {
    }

    @Override
    public void onDisconnect(Socket socket, String protocol) {
    }
}
```

```
    }  
  
    @Override  
    public void onMessage(String message, Socket socket, String protocol,  
        List connectedSockets) {  
        for(Socket sock : connectedSockets) {  
            try {  
                WebSocketMessage.sendMessage(message, sock);  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
  
    @Override  
    public void onMessage(byte[] message, Socket socket, String protocol,  
        List connectedSockets) {  
        // Binary message  
    }  
  
}
```

CAS Handler

Handler class name: *org.paw.handler.CasHandler*

Implements the CAS protocol: [CAS Protocol](#)

Parameters

prefix

URL protection prefix. Defaults to "/".

casServerUrl

Base CAS server URL.

cookieName

Name of the session cookie.

The default cookie name is "cookie".

A cookie must be available, it is not created by this handler.

To generate a session cookie a *sunlabs.brazil.handler.CookieSessionHandler* handler should be used prior to this handler inside the handler chain.