Device Capabilities

Introduction

Rhodes provide access to device specific capabilities such as GPS, PIM, camera, SMS, video player, accelerometer, proximity detector and **native UI elements**. Below is the device support matrix showing what release supports what device capabilities on a per device operating system basis.

Capability	iOS	Windows Mobile	Windows Phone	BlackBerry	Android	RhoSimulator	Symbian	Windows Desktop
GeoLocation	0.3	0.3	TBD	0.3	1.0	3.5	TBD	TBD
PIM Contacts	0.3	0.3	TBD	0.3	1.0	3.5	3.1	TBD
PIM Calendar	2.2	2.2	TBD	2.2	2.2	3.5	TBD	TBD
Camera	1.0	1.0	TBD	1.0	1.0	3.0	TBD	TBD
Barcode	2.1	2.1	TBD	2.1	2.1	4.0	TBD	TBD
Date/Time picker	1.2.2	2.0	TBD	1.2	1.2	3.0	TBD	3.0
Menu	1.2.2	2.0	3.0	1.2	1.5	3.0	3.1	3.0
Toolbar	1.2.2	2.3	3.0	n/a	1.5	3.0	3.1	3.0
Tab Bar	1.2.2	3.5	3.0	n/a	1.5	3.1	3.1	3.4
Nav Bar	2.0	3.5	TBD	n/a	2.0	4.0	TBD	TBD
Signature Capture	2.1	3.0	TBD	3.5	2.1	4.0	TBD	TBD
Audio/Video capture	4.0	4.0	TBD	4.0	4.0	4.0	TBD	TBD
Bluetooth	2.2	2.2	TBD	2.2	2.2	4.0	TBD	TBD
NFC	TBD	TBD	TBD	TBD	3.0	4.0	TBD	TBD
Push	1.2	4.0	TBD	1.2	2.2	3.5	TBD	TBD
Screen rotation	2.1	3.5	TBD	2.0	2.1	3.5	TBD	TBD
Native Maps	1.4	3.5	TBD	1.4	1.5	3.5	TBD	TBD
Alerts/Audio File Playback	1.2	1.5	TBD	1.2	1.2	3.1	TBD	3.1
Ringtones	3.5	1.5	TBD	1.5	1.5	3.5	TBD	TBD
Printing	3.4	3.4	TBD	TBD	TBD	TBD	TBD	TBD

As of Rhodes version 3.3.3, the Barcode, NFC, and Inline Signature Capture APIs, as well as Rhom data encryption are removed from Rhodes. These features are only supported in Motorola RhoMobile Suite. If you wish to use

these features, you will need to upgrade to RhoMobile Suite. Your application's build.yml will also need to be modified to indicate the application type is 'Rhoelements'. Additionally, a RhoElements license is required.

System class

The System Ruby class provides access to the system specific information.

To get value of a named system property:

```
System.get_property(property)
```

Properties:

- platform – APPLE, Blackberry, WINDOWS, ANDROID, SYMBIAN
- has_camera – is camera available on the device
- screen_width – screen width in logical pixels (used for set coordinates)
- screen_height – screen height in logical pixels (used for set coordinates)
- real_screen_width – screen width in real pixels
- real_screen_height – screen height in real pixels
- screen_orientation – screen orientation 'portrait', 'landscape'
- ppi_x – horizontal PPI (Pixels Per Inch)
- ppi_y – vertical PPI (Pixels Per Inch)
- has_network – is device connected to the network
- has_cell_network – is device connected to the cell network
- has_wifi_network – is device connected to the wifi network
- phone_number – device phone number
- device_owner_email – primary email of phone owner (works only on Android)
- device_owner_device_owner_name – name(account name) of phone owner (works only on Android)
- device_id – returns device ID which may be used to receive push messages. This is not phone device id, this is PUSH device id – you should register you phone for PUSH before get this parameter. Read **Push Notification documentation** about it. On iPhone it may be empty right after application started while Rhodes registers to receive PUSH notifications; you should wait till it become non-empty.
- phone_id – returns hardware based id. It depends on capabilities configured for an application and has to remain same even across application uninstall/install.
- full_browser – is full browser in the WebView control on Blackberry (see **here** for more details)
- device_name – name of device application running on. Examples: '9000' (BB), 'iPhone', 'dream' (Android)
- os_version – version of OS of device. Examples: '4.6.0.192' (BB), '3.0' (iPhone), '1.6' (Android)
- locale – current language like 'en', 'ru' etc.
- country – country abbreviation
- is_emulator – return true if application is running on emulator
- has_calendar – return true if calendar support is available

Get notifications on screen rotation:

```
System.set_screen_rotation_notification(callback_url, params)
```

- callback_url will be called when screen rotate; callback parameters will contain params you passed while setting up notification as well as `width`, `height`, and `degrees`

Exit application

```
System.exit
```

Enable\disable phone sleep by timeout, return previous state

```
System.set_sleeping
```

Open application associated with url; behavior may be different on different platforms and depend on installed software.

```
System.open_url(url)
```

> **On iPhone, if you post path of local file, then the specific iOS control (UIDocumentInteractionController) where file will previewed and user will be able open this file by selected application wich supported this filetype.**

Run specified application. Note: appname is platform dependent

```
System.run_app(appname, params)
```

> **On iPhone, appname is the same with registered application URL scheme. Params just string. Example : System.run_app('myapp', 'someparams'). Also this the same with : System.open_url('myapp:someparams')**

Install application from specified url

```
System.app_install(url)
```

Is specified application installed on device?

```
System.app_installed?(appname)
```

> **On iPhone, appname is the same with registered application URL scheme.**

Uninstall specified application

```
System.app_uninstall(appname)
```

Unzip file to the same folder where zip file is

```
System.unzip_file(local_path_to_zip)
```

Return command line parameters as a string. Worked only on iPhone now. Resetting when application go to background. If your URLBundle is "myapp" and you open application by url: "myapp:/app/page1", then start params will "/app/page1".

```
System.get_start_params()
```

Set application icon badge (number) – only for iPhone. set badge to 0 (zero) for remove badge.

```
System.set_application_icon_badge(badge_number)
```

Switch application locale at runtime:

```
System::set_locale('es') #set current locale to Spanish
```

This call does not change OS Locale, it will change only LocalizationSimplified locale used in helper methods. See **Localization** for more information.

To get current application Locale:

```
LocalizationSimplified::get_cur_locale #get current application locale
```

Sample

See layout.erb of **System API Sample application** for some of the examples of how to use System class. There is also **system test** which demonstrate System class usage.

Localization

See **Rhodes System API Samples application** as an example.

Rhodes use localization_simplified library to support non-English languages.

Add to build.yml:

```
extensions: ["rholang"]
```

In case of several extensions, insert space after extension name and comma:

```
extensions: ["rholang", "net-http"]
```

Create utf-8 encoded file in app `<app_folder>/app/lang/lang_<lang_id>_<country_id>.rb` or `<app_folder>/app/lang/lang_<lang_id>.rb`. This file will be automatically loaded by rhodes based on current locale.

For Example create lang_en.rb:

```
module Localization
    Views = {
        :greeting => "This is test"
    }
end
```

And use this string in the view:

```
<ul id="home">
    <li><%= Localization::Views[:greeting] %></li>
</ul>
```

To switch locale at runtime use:

```
System::set_locale('es') #set current locale to Spanish
```

Details:

All non-ascii symbols should be utf-8 encoded

To get current locale on the phone use System.get_locale method. It returns ‘en’, ‘de’ etc locale id’s

To show localized Date and Time:

```
Time.now.to_formatted_s(:long)
Time.now.strftime("%B %d, %Y %H:%M") # all names will be localized
Date.today.to_formatted_s(:long)
Date.today.strftime("%B %e, %Y") # all names will be localized
```

To show currency (see rails analog for details):

```
Rho::NumberHelper.number_to_currency
```

Geolocation

Geolocation information is available in two ways: as an asynchronous service through Ajax calls to a predefined local URL, or in a controller using Ruby calls to the GeoLocation class.

You need to enable the GPS capability to use GeoLocaiton. This is done by adding the following line to build.yml:

```
capabilities:
  - gps
```

See the **Configuration** section for more details.

The GPS receiver consumes a significant amount of energy, and should be switched off most of the time in order to preserve the device's battery life. Any call to GeoLocation method or notification call back will power up the GPS receiver. The GPS receiver will be switched off automatically after some time (see **Configuration** for parameters) or explicitly by GeoLocation.turnoff. Also note that it usually takes a while to acquire the device's current position; in some cases, it may take several minutes and may not work at all indoors.

Note that in old rhodes versions (up to 2.3.1) this feature uses only GPS location provider. In more recent rhodes, it also uses network based location determination. Network based location is not as precise as GPS, but in most cases it consumes less power and returns results faster than GPS. Rhodes will update location using network data until the GPS signal has been received; then only GPS will be used for more precise data.

There are two ways to use GeoLocation.

- Synchronous call to GeoLocation module for particular value.
- Set GeoLocation notification (geoLocation.set_notification) and track location by the notification callback. In this case all location values are passed to notification as parameters in a @param variable.

Note that GeoLocation is switched off automatically when the application goes into the background, and is switched on as the application goes back to foreground. However it is possible to switch on location services as usual while the application is in background.

See the **Rhodes-System-Api-Samples example** application for GeoLocation usage example.

Asynchronous Ajax calls

The Rhodes framework provides a simple HTML tag for accessing Geolocation information. To use it, include the appropriate javascript library on your page:

- For iPhone, Android: `/public/jquery/jquery-1.6.2.min.js` and `/public/js/rhogeolocation.js`
- For Windows Mobile: `/public/js/rhogeolocation-wm.js`
- For BlackBerry: Unsupported. The BlackBerry webview control doesn't support Ajax.

Then add one of the following tags in the appropriate location in your HTML: , or . The included javascript will query a predefined URL and fill these tags with location information.

`<geolocation/>` – returns a string in the form `[formatted position];[latitude];[longitude]`.

For example: 37.3317° North, 122.0307° West;37.331689;-122.030731

`<geolatitude/>` – returns just the latitude
`<geolongitude/>` – returns just the longitude

GeoLocation Ruby class

Get current latitude:

```
GeoLocation.latitude
```

Get current longitude:

```
GeoLocation.longitude
```

Test if the location system is up and acquired position:

```
GeoLocation.known_position?
```

Set callback to track location changes.

```
GeoLocation.set_notification(
    callback, callback_param="", ping_gpssystem_interval=0)
```

- callback – url for the notification callback; after the callback has been called once, it will automatically be called repeatedly with the current location coordinates passed to the callback in the @params variable. (You need not reset the callback within the callback.)
- callback_param: this string will be added to the body of the callback url. You can use it to identity who is setting up the callback, such as “my_tag=55”. In general you do not set callback_param (leave it blank as in “”).
- ping_gpssystem_interval – Optional. If 0, the system interval is used; the callback is executed when the GPS system processes a location update (dependent on the mobile platform). If set to a number (such as 3), the callback is executed at an interval of this number of seconds (such as every three seconds).

When the GeoLocation.set_notification callback is called, it will receive a variable called @params, just like a normal Rhodes controller action. Here are the parameters included in the @params variable.

- known_position – 1 or 0
- latitude
- longitude
- available – 1 or 0. This is the availability of GeoLocation: not only does the hardware exist, but also the user can turn GPS off in phone settings, or not allow GPS activity on iPhone, etc.
- status – ‘error’ or ‘ok’
- error_code – from RhoError.
- accuracy – horizontal radius in meters; iOS and Android.

> **You only need to call GeoLocation.set_notification once. The current behavior of the callback is that it will be called forever until it is stopped; you need to call `GeoLocation.turnoff` to stop it. The previous behavior was that the callback was called once and needed to be reset.**

> **We do not have a timeout parameter to automatically turn off the GPS system. If you want to turn off the GPS system, call GeoLocation.turnoff.**

Turn off GeoLocation.

```
GeoLocation.turnoff
```

> **When you call GeoLocation.turnoff, after the GPS is switched off, you might still receive a few callbacks (this depends on the platform; iOS and Android does not receive callbacks after turnoff).**

Get the distance between two points in miles.

```
```

```
GeoLocation.haversine_distance(
    latitude1, longitude1, latitude2, longitude2)
```

GeoCoding: You can use any free web service for direct and reverse geocoding. See our complete example of using Google geocoding web service in **Rhodes-System-Api-Samples example**.

> **On iOS and Android, real GPS starts working after the first access to the GeoLocation module. Real GPS hardware is switched off after a call to GeoLocation.turnoff. But keep in mind – any call to GeoLocation, such as getting latitude, starts the hardware GPS again!**

Testing GeoLocation

While developing your application on Windows Mobile emulator, you may find the **FakeGPS** utility useful.

Before testing on BB simulator, select menu Simulate/GPS Location and set your position.

To provide Mock Location Data on Android, see **here**

Samples

Tracking location all the time

You may decide to keep track of your position right after application starts. To do that, add following to your application.rb:

```
class AppApplication < Rho::RhoApplication
  def on_activate_app
    #start geolocation
    GeoLocation.set_notification("/app/Settings/geo_callback", "", 3)
  end
end
```

Callback points to the `geo_callback` in the Settings controller.rb:

```
class SettingsController < Rho::RhoController
  def geo_callback
    puts "geo_callback : #{@params}"
    # do something on position changes
    #...
  end
end
```

Requiring location only on a specific view

```
def show_location
  # check if we know our position
  if !GeoLocation.known_position?
    # wait till GPS receiver acquire position
    GeoLocation.set_notification( url_for(:action => :geo_callback), "")
    redirect url_for(:action => :wait)
  else
    # show position
    render
  end
end

def geo_callback
  # navigate to `show_location` page if GPS receiver acquire position
  if @params['known_position'].to_i != 0 && @params['status'] =='ok'
```

```
      GeoLocation.set_notification '', '', 2
      WebView.navigate url_for(:action => :map_all)
    end
  end
```

Turning off GeoLocation as soon as the app goes off the device front page

In this code sample, /GeoLocation/ has to be replaced with an appropriate regular expression to detect the controller. This code is called from geo_callback. This decision allows you to switch off GeoLocation from a single code point in case geo_callback is set.

```
def geo_callback
  puts "geo_callback : #{@params}"

  if WebView.current_location !~ /GeoLocation/
    puts "Stopping geo location since we are away of geo page: " + WebVie
    GeoLocation.turnoff
    return
  end
end
```

If the app does not need location updates, and GeoLocation should still remain active, you can send an empty notification.

```
GeoLocation.set_notification "", "", 30
```

Code Sample

See controller and view in the /app/GeoLocation folder of the **System API Samples application** for more information.

PIM Contacts

Rhodes provides access to the device's local phone book and stored contacts via the Ruby class RhoContact.

To allow read/modify personal information and contacts enable the pim capability. This is done by adding the following lines to build.yml:

```
capabilities:
  - pim
```

The following methods are available in the RhoContact class:

Return hash of hashes of contacts stored in the phonebook (index):

```
Rho::RhoContact.find(:all)
```

On some platforms an extended Rho::RhoContact.find routine is implemented

```
Rho::RhoContact.find(:all, params)
Rho::RhoContact.find(:first, params)
Rho::RhoContact.find(:count, params)
```

Currently the extended functionality is available at Android and iOS

- :all – get all contacts as usual
- :first – find first contact from list

- :count – returns count of contacts

For these selectors an additional params hash may be used. The next keys and values are recognized in additional params:

- :per_page – max count if contacts returned by find
- :offset – offset from begin of contacts' list These two parameters are used to paginate contacts list. If used with :count exact number of returned contacts can be determined (for example for last page).

:max_results – used with :count. maximum number of contacts to be returned.

The next parameters are available at Android only.

- :select – list of contact properties have to be selected from phonebook (Use it to enchance query performance)
- :conditions – hash of conditions to query contacts. Look for more details below.

> **It is allowed to pass additional params hash at all platforms. Platforms that has no extended functionality will just skip these**

Return hash of all properties of the contact identified by the provided id (show):

```
Rho::RhoContact.find(@params['id'])
```

Create new contact in the phonebook, set properties of the contact passed as parameter hash, and save created phonebook record (create). Return the new contact as a hash.

```
created_contact = Rho::RhoContact.create!(@params['contact'])
```

Find contact record in the phonebook, update record properties from the hash passed as parameter, and save updated record. Contact id passed in the hash (update):

```
Rho::RhoContact.update_attributes(@params['contact'])
```

Remove contact identified by the provided id from the phonebook. (delete)

```
Rho::RhoContact.destroy(@params['id'])
```

On all devices, properties currently supported are:
“id”,“first_name”,“last_name”,“mobile_number”,“home_number”,“business_number”,“email_address”,“company_name”

iPhone

On iPhone, additional contact properties are supported.

General: “prefix”, “first_name”, “middle_name”, “last_name”, “suffix”, “nickname”, “birthday”, “anniversary”, “created”, “updated”, “company_name”, “job_title”, “assistant_name”, “assistant_number”, “spouse_name”, “person_note”

> **“birthday”, “anniversary”, “created”, “updated” properties expect a date formatted as YYYY-MM-DD**

Addresses: “street_address_1”, “city_1”, “state_1”, “zip_1”, “country_1” “street_address_2”, “city_2”, “state_2”, “zip_2”, “country_2” “street_address_3”, “city_3”, “state_3”, “zip_3”, “country_3”

> **Address 1 is mapped to “work,” 2 is to “home,” 3 is to “other.”**

Email addresses: “email_address”, “home_email_address”, “other_email_address”

> **“email_address” is mapped to “work”**

Phone numbers: “business_number”, “home_number”, “mobile_number”, “main_number”, “pager_number”, “home_fax”, “work_fax”

Home pages: “home_page”

Android

> **In most cases the result of RhoContact::find query may return more contact properties then defined in :select**

On Android the next contact properties are supported (every of them may be used in :select list):

General: “id”, “display_name”, “first_name”, “last_name” The “display_name” is always filled by Android even if “first_name” and “last_name” are both empty. In this case email or phone number may be used.

Phone numbers: “mobile_number”, “home_number”, “business_number”

E-mails: “email_address”

Company: “company_name”

Conditions

:conditions parameter is a hash of conditions. Keys of the hash are contact property selectors (condition to be applied to) and values are conditions itself. The next property selectors are defined: – :phone – :email A property selector has no one-to-one relation to a single contact property like mobile_phone or email_address. Selectors are mapped to whole property group like all phones or all emails (support for several emails may be implemented in future). Condition like

```
{:phone => 'not_nil'}
```

will select all contacts which has at least one phone. The next conditions are currently suported: – ‘not_nil’ – ‘is_nil’

> **It may be useful to sort received contacts (espessially if paginated). Contacts is ordered by ‘display_name’ column and then splitted for pages. Unfortunately the order inside page is lost while passing the list to ruby**

```
@count = Rho::RhoContact.find(:count, :conditions => {:phone => 'not_nil'
if @params['offset']
    @offset = @params['offset'].to_i
else
    @offset = 0;
end
@contacts = Rho::RhoContact.find(:all, :per_page => 10, :offset => @offse
@contacts = {} unless @contacts
@contacts = @contacts.sort do |x,y|
    res = 1 if x[1]['display_name'].nil?
    res = -1 if y[1]['display_name'].nil?
    res = x[1]['display_name'] <=> y[1]['display_name'] unless res
    res
end
```

Sample

For examples on how to use the API provided by this class, see the view and controller in the /app/Contacts folder in the **System API Samples application**.

PIM Calendar/Events

Rhodes provides access to the device's local calendar and stored events via the Ruby class RhoEvent.

To allow read/modify calendar information enable the calendar capability. This is done by adding the following lines to build.yml:

```
capabilities:
  - calendar
```

Check if the device has a calendar:

```
System::get_property('has_calendar')
```

The following methods are available in the RhoEvent class:

Return hash of hashes of all events stored in the calendar (index):

```
Rho::RhoEvent.find(:all)
```

Return hash of all properties of the event identified by the provided id (show):

```
Rho::RhoEvent.find(@params['id'])
```

Return hash of all properties of the events found by specified parameters (index):

```
Rho::RhoEvent.find(:all, :start_date=>start,
    :end_date => endtime, :find_type=>'starting', :include_repeating => t
```

- start_date, end_date – define range where at least one of the Event's occurrences falls, inclusive.
- find_type – determines the criteria for matching an event occurrence;
    - 'starting' searches for events starting between start_date and end_date,
    - 'ending' searches for events ending between start_date and end_date,
    - 'occurring' searches for events that have any part of the event occurring during the period specified by start_date and end_date.
- include_repeating – if false then only search based on an Event's START and END values and do not calculate repeating occurrences of the event; if true then repeating occurrences of an Event are included during the search.

Create new event in the calendar, set properties of the event(excluding 'id', which is generated) from passed as parameter hash, and save created calendar event (create), also return hash of all properties of the created event (include 'id' property):

```
created_event = Rho::RhoEvent.create!(@params['event'])
```

Find event in the calendar, update record properties from the has passed as parameter and save updated event. Event id passed in the hash (update):

```
Rho::RhoEvent.update_attributes(@params['event'])
```

Rho::RhoEvent.destroy(@params['id']) #=> remove event identified by the provided id from the calendar. (delete)

On all devices, properties currently supported are:
"id","title","location","notes","start_date","end_date"

On Blackberry and Windows mobile additional contact properties are supported. These are: “reminder”(in minutes), “privacy”(‘public’, ‘private’, ‘confidential’)

Recurrence properties: “recurrence”, “frequency”(‘daily’, ‘weekly’, ‘monthly’, ‘yearly’), “interval”, “end_date” , “days”(array of 7 items, 1 – means event is triggered – Mon-Sun), “months” (array of 12 items, Jan-Dec), “weeks”(array of 5 items, first-fifth), “count”(how many occurrences), “day_of_month”(1..31)

On iPhone and Android additional event properties are supported. These are: “canceled”, “organizer”, “attendees”, “last_modified”,

Recurrence properties: “recurrence”, “frequency”(‘daily’, ‘weekly’, ‘monthly’, ‘yearly’), “interval”, “count”, “end_date” “count” and “end” rucurrence properties are mutual exclusive. “end” is date/time beyond last event occurence but within same day.

Recurrence properties

Recurrence properties are groupped in separate hash stored as single event property (event[Rho::RhoEvent::RECURRENCE]).

Names of properties

There are helper constants defined in Rho::RhoEvent which can be used to address event values and as predefined set of values (for frequency).

- ID
- TITLE
- CANCELED
- ORGANIZER
- START_DATE
- END_DATE
- LAST_MODIFIED
- LOCATION
- NOTES
- PRIVACY
- REMINDER
- RECURRENCE
- RECURRENCE_FREQUENCY
- RECURRENCE_FREQUENCY_DAILY
- RECURRENCE_FREQUENCY_WEEKLY
- RECURRENCE_FREQUENCY_MONTHLY
- RECURRENCE_FREQUENCY_YEARLY
- RECURRENCE_INTERVAL
- RECURRENCE_COUNT
- RECURRENCE_END
- RECURRENCE_DAYOFMONTH
- RECURRENCE_MONTHS
- RECURRENCE_DAYS

Sample

For examples on how to use the API provided by this class, see the view and controller in the /app/Calendar folder in the **System API Samples application**.

Camera

You need to enable the Camera capability. This is done by adding the following lines to build.yml:

```
capabilities:
  - camera
```

Main API

Check if the device has a camera:

```
System::get_property('has_camera')
```

The Camera API provides the following functionality:

Take a picture:

```
Camera::take_picture('/app/model/camera_callback')
```

Choose a picture from an album:

```
Camera::choose_picture('/app/model/camera_callback')
```

Once the user has taken/chosen a picture, the callback URL you specified will be called. The callback is a POST message; the body of the message contains the return status and image_uri.

- Status is ‘ok’, ‘cancel’, or ‘error’
- image_uri points to the taken/chosen image stored in the /public/db-files folder; the image file will have auto-generated name.

Extended API (only on iOS and Android)

Take a picture:

```
Camera::take_picture('/app/model/camera_callback', options_hash)
```

Choose a picture from Image Gallery (currently supported only on iPhone):

```
Camera::choose_picture('/app/model/camera_callback', options_hash)
```

options_has – hash contain additional parameters :

- “camera_type” – “default”(is the same with “main”), “main”, “front”. Default value = “default”
- “desired_width” – number, desired width in pixels. Default value = max camera width
- “desired_height” – number, desired height in pixels. Default value = max camera height
- “color_model” “RGB”, “Grayscale”. Default value = “RGB”
- “format” “jpg”, “png”. Default value = “jpg” If you do not define this property when use choose_picture, then on iOS type of image in Gallery will recognize and use the same (JPG or PNG) for save image to applications data.
- “enable_editing” – boolean, enable post photo capture image customize (supported only on iPhone !). Default value = true. If you enable this, image will captured reduced to screen size (not full size)!
- “flash_mode” – string, supported only on Android ! Values : “off”, “on”, “auto”, “red-eye”, “torch”.
- “save_to_shared_gallery” – boolean, supported for iOS. if true, picture you take will be added to the camera roll. Default value = false.

Additional parameters in camera callback :

- “status” – “ok”, “cancel”, “error”
- “message” – if status == error, then in this value contain message about error
- “image_uri” – uri of saved image file
- “image_width” – image width
- “image_height” – image height
- “image_format” – “png”, “jpg”

Get camera info:

```
Camera::get_camera_info(camera_type)
```

camera_type – “default”(is the same with “main”), “main”, “front” return Hash with camera info for selected camera type or NIL if this type is not supported. In returned hash:

- “max_resolution” – hash with “width” and “height” numeric values.

Sample

See controller and view in the /app/Image folder of the **System API Samples application** for more information.

Barcode

> **As of Rhodes version 3.3.3, the Barcode API is removed from Rhodes. This feature is only supported in Motorola RhoMobile Suite. If you wish to use this feature, you will need to upgrade to RhoMobile Suite. Your application's build.yml will also need to be modified to indicate the application type is 'Rhoelements'. Additionally, a RhoElements license is required.**

The Barcode API allows to try recognize barcode on an image:

```
Barcode.barcode_recognize(image_file_full_path)
```

Barcode recognition functionality realized as Rhode Native extension. You should add "Barcode" to extension list in build.yml located in your application folder.
`Barcode.barcode_recognize(image_file_full_path)` return string with recognized code or emty string if not any recognized barcodes on the image. You can combine Camera for get picture with Barcode recognition for process barcode data – see sample below.

For barcode recognition we use Zbar library (iPhone, and Windows Mobile platforms) and ZXing library (Android and BlackBerry platforms). In this case we support next barcode types :

- WM platform: UPC-A, UPC-E, EAN-8, EAN-13, Code 39, Code 128 .
- iPhone platform: UPC-A, UPC-E, EAN-8, EAN-13, Code 39, Code 128, QR Code .
- Android and BlackBerry platforms: UPC-A, UPC-E, EAN-8, EAN-13, Code 39, Code 128, Code 93, QR Code, ITF, Codabar

> **We are very sorry, but we are removed ZBar source code from our project by ZBar license requirements. For build Barcode extension for WM, you should download ZBar sources from http://zbar.sourceforge.net/ and copy to [rhodes root]/lib/extensions/barcode/ext/barcode/shared/zbar/zbar/ folder "include" and "zbar" folders from ZBar sources. Also you should fix ZBar sources for building by Microsoft Visual Studio (move define of inner function variables to begin of functions, etc.). See detailed instructon in your Rhodes installation at /lib/extensions/barcode/ext/barcode/shared/zbar/zbar/README.TXT**

On Android and iOS (not supported on iPhone 2G and iPhone 3G !) now supported real-time barcode recognizing (second parameter is hash with options):

```
Barcode.take_barcode(url_for(:action => :take_callback), {})
```

On Android also you can use front camera for take_barcode :

```
Barcode.take_barcode(url_for(:action => :take_callback), {:camera => 'fro
```

This code open special UI for real-time barcode recognizing from camera. If any barcode found recognizing stopped and user can select – confirm recognizing barcode or continue recognizig. If user confirmed or cancelled the callback will be called. In callback are these parameters.

- "status" – "ok" or "cancel"
- "barcode" – recognized barcode string

Sample

See controller and view in the /app/BarcodeRecognizer folder of the **System API Samples application** for more information.

Signature Capture

The Signature Capture API allow take a signature and save it as an image:

```
Rho::SignatureCapture.take(callback_url, params)
```

Signature Capture open window for hand writing, where user draw his signature. Signature saved to an image file. You can choice 'jpg' or 'png' format for image.

- callback_url – callback url
- params – hash with params

Example:

```
Rho::SignatureCapture.take(url_for( :action => :signature_callback), { :i
```

The following parameters are in the hash.

- imageFormat – "jpg", "png" or "bmp"
- penColor – color of pen used for drawing signature
- penWidth – wide of pen
- bgColor – color of background

> **On Windows Mobile and Windows CE devices only "bmp" is supported as 'imageFormat'.**

After user entered signature (or canceled) provided callback url will be called. Callback parameters will be the following.

- status – 'ok' or 'cancel'
- signature_uri – path to the image with signature

Inline signature capture

> **As of Rhodes version 3.3.3, the Inline Signature Capture API is removed from Rhodes. This feature is only supported in Motorola RhoMobile Suite. If you wish to use this feature, you will need to upgrade to RhoMobile Suite. Your application's build.yml will also need to be modified to indicate the application type is 'Rhoelements'. Additionally, a RhoElements license is required.**

Windows Mobile and Windows CE devices support signature capture started in window, which you can show over current page(scrolling is not supported in this case).

To display window hand writing, where user draw his signature call:

```
Rho::SignatureCapture.visible(visibility, params)
```

- visibility – true or false
- params – hash with params, same as for 'take'

Example :

```
Rho::SignatureCapture.visible(true, { :imageFormat => "jpg", :penColor =>
```

To save Signature to an image file and call callback:

```
Rho::SignatureCapture.capture(callback_url)
```

- callback_url – callback url

Example :

```
Rho::SignatureCapture.capture(url_for( :action => :signature_callback))
```

To clear Signature in the window:

```
Rho::SignatureCapture.clear()
```

Example :

```
Rho::SignatureCapture.clear()
```

Sample

See controller and view in the /app/SignatureUtil folder of the **System API Samples application** for more information.

Bluetooth

Bluetooth API provide access to Bluetooth serial port connection between phone and another phone, phone and PC, phone and external Bluetooth device (for example external Bluetooth GPS device).

To allow Bluetooth enable the bluetooth capability. This is done by adding the following lines to build.yml:

```
capabilities:
  - bluetooth
```

Currently Bluetooth support have following limitations:

- on Apple device (iPhone, iPad etc.) you can connect only to another Apple device
- on Blackberry if you want to make client connection to any device, you should pair devices only when on another device running server service, because it need for add paired device to list of devices with support serial port profile – only that devices you can see in the list of devices to connect
- on any platform, except iPhone, you should pair you devices before make connection
- you can change local device name for display on another device only on iOS and Android platform – for WM and BB use system settings for change local device name
- you can not see another device name, when you using server connection on BlackBerry
- only one connection session can created and using on any device in this time. You should close current session before make another session.
- one-to-few peers connection scheme is unsupported now.

There are two steps to make connection and start using of Bluetooth :

- Make connection session: execute `Rho::BluetoothManager.create_session` and setup callback where you receive result of connection. That callback will get events related to making connection(connect ok, canceled by user, error). Connection provided by platform specific UI.
- For make connection without UI, you should execute `Rho::BluetoothManager.create_server_and_wait_for_connection` on server and `Rho::BluetoothManager.create_client_connection_to_device` on client. On client you should specify server name(display name – not Bluetooth ID!) for connect. Specify callback – callback have the same parameters with `Rho::BluetoothManager.create_session`. You can cancel connection process by `Rho::BluetoothManager.stop_current_connection_process`
- After receiving successful result and name of connected device in create_session_callback you should setup session_callback by `Rho::BluetoothSession.set_callback` to process incoming session events from connected device (data received) or event related to this session connection(disconnect, errors).

Connection without UI worked only on iOS and Android platforms!

Bluetooth API

BluetoothManager

Get availability of Bluetooth on the device. Return true/false:

```
Rho::BluetoothManager.is_bluetooth_available
```

Switch Bluetooth off:

```
Rho::BluetoothManager.off_bluetooth
```

Change local device name:

```
Rho::BluetoothManager.set_device_name(name)
```

Get local device name for current device (this name displayed on another device)

```
Rho::BluetoothManager.get_device_name
```

Get last error. Returns OK/ERROR/CANCEL

```
Rho::BluetoothManager.get_last_error
```

Creates Bluetooth session. Return OK/ERROR:

```
Rho::BluetoothManager.create_session(role, callback_url)
```

- role – may be ROLE_SERVER or ROLE_CLIENT
- callback_url – this url will be called after session was created or canceled. Parameters received in the callback:
    - status – OK / ERROR / CANCEL
    - connected_device_name – name of the connected device

Connect without UI: make current device discoverable for other, wait for client connection from other device

```
Rho::BluetoothManager.create_server_and_wait_for_connection(callback_url)
```

- callback_url – this url will be called after session was created or canceled. Parameters received in the callback:
    - status – OK / ERROR / CANCEL
    - connected_device_name – name of the connected device

Connect without UI: make client connection to device:

```
Rho::BluetoothManager.create_client_connection_to_device(server_name, cal
```

- server_name – name of other device. If you make connect with UI, you receive that name as connected_device_name

- callback_url – this url will be called after session was created or canceled. Parameters received in the callback:
    - status – OK / ERROR / CANCEL
    - connected_device_name – name of the connected device

Connect without UI: cancel current connection:

```
Rho::BluetoothManager.stop_current_connection_process
```

BluetoothSession

Set Bluetooth session callback

```
Rho::BluetoothSession.set_callback(
    connected_device_name, session_callback_url)
```

- connected_device_name – name of the connected device
- return OK/ERROR
- parameters in callback:
    - 'connected_device_name' – name of connected device
    - 'event_type' – SESSION_INPUT_DATA_RECEIVED / ERROR / SESSION_DISCONNECT

Disconnect from the device

```
Rho::BluetoothSession.disconnect(connected_device_name)
```

- connected_device_name – name of the connected device
- return OK/ERROR

Get session status

```
Rho::BluetoothSession.get_status(connected_device_name)
```

- return received but not read data size; -1 if error, 0 if empty(not actually received data)

Read data

```
Rho::BluetoothSession.read(connected_device_name)
```

- connected_device_name – name of the connected device
- return array of bytes

Write data

```
Rho::BluetoothSession.write(connected_device_name, data)
```

- connected_device_name – name of the connected device
- data must be array of byte/fixnum

Read string

```
Rho::BluetoothSession.read_string(connected_device_name)
```

- connected_device_name – name of the connected device
- return string

Write string

```
Rho::BluetoothSession.write_string(connected_device_name, data)
```

- connected_device_name – name of the connected device
- data must be a string
- return OK/ERROR

Example of Controller with using Bluetooth API

This is not a complete example (for a link to a complete example, see the link below this example). This code just shows how you can make a connection and send/receive strings.

```
require 'rho/rhocontroller'
require 'rho/rhobluetooth'

class BluetoothController < Rho::RhoController
  @layout = :simplelayout
  $connected_device = nil

  def index
    render
  end

  def start_bluetooth
    if Rho::BluetoothManager.is_bluetooth_available()
      Rho::BluetoothManager.create_session(Rho::BluetoothManager::ROLE_CL
    end
  end

  def send_string(str)
    Rho::BluetoothSession.write_string($connected_device, str)
  end

  def connection_callback
    if @params['status'] == Rho::BluetoothManager::OK
       $connected_device = @params['connected_device_name']
       Rho::BluetoothSession.set_callback($connected_device, url_for( :ac
       send_string('Hello friend !')
    end
  end

  def session_callback
     if @params['event_type'] == Rho::BluetoothSession::SESSION_INPUT_DAT
        while Rho::BluetoothSession.get_status($connected_device) > 0
          str = Rho::BluetoothSession.read_string($connected_device)

          # use received string

        end
     end
  end

  def close_all
    Rho::BluetoothSession.disconnect($connected_device)
    Rho::BluetoothManager.off_bluetooth()
  end

end
```

Example of chat application using Bluetooth connection

You can find complete example of using Bluetooth API in **Rhodes-System-Api-Samples**. See Bluetooth Chat Demo page – **BluetoothChat**. In this example you can see how to exchange text messages between two different devices. You also can use this example for connect to external

Bluetooth device (external GPS device for example) or PC or Mac (use terminal to see and send messages).

NFC

NFC (Near Field Communication). NFC API provide access to NFC functionality. You can check NFC availability on current device and register callback for listen event when NFC tag near of device. Currently NFC supported only on Android. And also Android version must be 2.3.3 or later. NFC API implemented in native extension. You should add “nfc” to extension list in your nuild.yml before start using NFC in your application. Please see detailed doc: **Rhodes NFC extension**.

> **As of Rhodes version 3.3.3, the NFC API is removed from Rhodes. This feature is only supported in Motorola RhoMobile Suite. If you wish to use this feature, you will need to upgrade to RhoMobile Suite. Your application’s build.yml will also need to be modified to indicate the application type is ‘Rhoelements’. Additionally, a RhoElements license is required.**

Ringtone manager

The Ringtone manager API provides access to view/play the user’s installed ringtones.

Get all available ringtones

```
@ringtones = RingtoneManager::get_all_ringtones
```

- The variable returned by get_all_ringtones will be a hash containing key/value pairs, where the key is the user-friendly name of ringtone, and the value is its full file name.

Play a given ringtone

```
RingtoneManager::play @ringtones['My Ringtone']
```

> **If ‘play’ is called while another ringtone is playing, it will stop the old ringtone and start the new one.**

Halt playing of a ringtone

```
RingtoneManager::stop
```

- Can safely be called even if no ringtone is playing

> **Currently implemented for Android, Blackberry and Windows mobile. On Blackberry, only the user installed ringtones are accessible. System preinstalled ringtones are not accessible due to Blackberry limitations.**

Sample

See controller and view in the /app/Ringtones folder of the **System API Samples application** for more information.

Printing

Printing on iOS

iOS inlude printing support. You should prepare File for printing and open it by standart platform – use:

```
System.open_url(file_full_path)
```

We recommend prepare PDF. You can use PDF-Writer pure Ruby library for it – we include it into

rhodes – just include “pdf-writer” and “thread” extensions to your application extension list in build.yml :

```
extensions: ["pdf-writer", "thread"]
```

See example in our **System API Samples application Generate PDF example** application.

Printing on Android

Android do not support printing. You can use any third-party application for printing, for example Google Cloud Print application for print by using of Google Cloud Print web service. You should prepare file for printing and open it by standart platform – use:

```
System.open_url(file_full_path)
```

Android platfrom open that file in application registered for file extension or show select dialog for manually choose application if there are more than one application registered for this extension.

We recommend prepare PDF. You can use PDF-Writer pure Ruby library for it – we include it into rhodes – just include “pdf-writer” and “thread” extensions to your application extension list in build.yml.

```
extensions: ["pdf-writer", "thread"]
```

See example in our **System API Samples application Generate PDF example** application.

PUSH Notifications

Push notification support is currently available for Android, Blackberry and iPhone.

Testing push is a little more involved than some other features because it requires additional setup on both the RhoConnect application and Rhodes application. Please refer to **RhoConnect Device Push** for more information.

Setup

First, enable push in your rhodes application in build.yml:

```
capabilities:
  - push
  - vibrate #=> if you want to enable vibrate in your push messages
```

Next, your RhoConnect application should be running on a network that is accessible to your device. For example, you might have your RhoConnect application running on a LAN with IP 192.168.1.10, PORT 9292. Then make sure your device is connected to the same LAN and setup your application **rhoconfig.txt** syncserver as follows:

```
syncserver = 'http://192.168.1.10:9292/application'
```

Now your Rhodes application is configured to receive push messages.

To handle push message in ruby code it’s needed to register push callback

```
class AppApplication < Rho::RhoApplication
  def initialize
    super
    System.set_push_notification "/app/Push/push_callback", ""
  end
end
```

Push callback set up in form of local application URL and called as callback nevermind rho_callback=1 specified in it's parameters list or not. There is several push message parameters that can be handled by Rhodes authomatically. These are: – alert – sound – vibrate – do_sync Every parameter can be specified in a push message body with its value:

```
alert=some message here&sound=alert.wav&vibrate=3&do_sync=sync_src1,sync_
```

Automatic push message parameters handling takes its place in case no custom push callback or push callback returns "rho_push":

```
def push_callback

  # do some work here

  "rho_push"
end
```

iPhone

iPhone PUSH support uses the Apple Push Notification Service (APNS) introduced in iPhone SDK 3.0. In order to use the service, you will need to obtain a push-enabled provisioning profile and a server-side push certificate used for encrypting RhoConnect->APNS traffic.

In case of iPhone, regardless of the operation, the user will be presented with the option to activate the application if it is not running.

> **For testing push, you will need to build and deploy your application to a physical iPhone (or iPad) device. Apple does not support testing push on simulators.**

Setup Push Certificate

The first step to setting up your environment is to create an APNS push certificate. To create the SSL certificate, follow the "Creating the SSL Certificate and Keys" section on the **developer site**.

Once you have the certificate, you will need to export it to a .pem format using **these instructions**. Make sure to name the output file 'apple_push_cert.pem' file from these instructions, since this is what we configured in our RhoConnect application.

Once you have this file, copy it to your RhoConnect application directory under the settings folder:

```
$ cd myrhoconnectapp
$ cp /path/to/apple_push_cert.pem settings/apple_push_cert.pem
```

Now your RhoConnect application is ready to send APNS messages.

Setup Provisioning Profile

Next, you can setup your Apple Push Provisioning Profile using Apple's **developer site instructions**. Once you have the profile installed in XCode and on your test device, you are ready to test push.

Setup iPhone Device

Setting up the device is the same process as **building any normal Rhodes application**. When you start the application, make sure to also open the console window (cmd-shift-R). When the application starts, you should see some output in the console that shows the device token (towards the top):

```
2010-08-19 10:14:22.627 rhorunner[1486:307] Device token is <10fd92ab fa8
```

This confirms that your application is running with push enabled. Once you login to the RhoConnect application and sync, you will see the device registered on the RhoConnect console under the user id you used to login. Now you are ready to **test push from your RhoConnect application**.

Alert Audio File Setup for iPhone

In case of iPhone, audio files for the push alert should be placed in the `/public/alerts` folder and build script will copy them into root of the application main bundle (iPhone wouldn't play file from any other place).

Android

Android PUSH support uses the Android Cloud to Device Messaging (AC2DM) system introduced in Android 2.2 (Froyo). In order to use the service, you will need to register your role-based google account (or use existing one) and then register this account in **Google C2DM program**.

Here are some guidelines for developing and testing an Android application that uses the C2DM feature:

- To develop and test your C2DM applications, you need to run and debug the applications on an Android 2.2 system image that includes the necessary underlying Google services.
- To develop and debug on an actual device, you need a device running an Android 2.2 system image that includes the Market application.
- To develop and test on the Android Emulator, you need to download the Android 2.2 version of the Google APIs Add-On into your SDK using the Android SDK and AVD Manager. Specifically, you need to download the component named "Google APIs by Google Inc, Android API 8". Then, you need to set up an AVD that uses that system image.
- Android C2DM system uses an existing connection for Google services. This requires users to set up their Google account on their mobile devices (and on emulator!).

Setup application

As first step, register role-based google account for your application. Here for example it will be referred as push-app@gmail.com.

Then modify your application's build.yml and specify there google account used to send PUSH messages:

```
android:
  push:
    sender: push-app@gmail.com
```

This is the same address used by RhoConnect to retrieve authtoken. See **here** for more information.

> **WARNING! This e-mail address MUST NOT be the same as origin of google account on phone! This mean that google account registered on your phone/emulator MUST NOT be push-app@gmail.com; it MUST be another one. This is known bug in Android C2DM implementation – if sender and receiver are the same, underlying android service crash and lose message. More details here. This is actual not for all Android versions.**

At Android it is possible to set up status bar notifications for PUSH messages. In this case push message is handled by application only after user has opened the notification. There are two modes available for push notifications: – always – push notification is shown always – background – push notification is shown only if application is backgrounded (not started) If no push notification mode is specified in build.yml then no notifications are shown at all.

```
android:
  push:
    notification: background
```

BlackBerry

Notifications to BlackBerry are sent using PAP 2.0 message through a BES/MDS server.

On the simulator, this is done via the MDS simulator tool (which you've probably already seen in the form of a console window everytime you do 'rake run:bb'). On the device, this is done through the BES/MDS server that the phone is configured to use.

These instructions assume you are familiar with BES/MDS concepts (for more information please see **here**).

In case of Blackberry, if the application is in the background, an alert operation will bring the application upfront; other operations will not.

Setup MDS Simulator

Make sure you close the BlackBerry simulator and MDS simulator before continuing for your changes to take effect!

To enable the push port in your MDS simulator, edit the following file:

```
C:\Program Files\Research In Motion\BlackBerry JDE <VERSION YOU ARE BUILD
```

Uncomment the last line of the file, which should be 'push.application.reliable.ports=100':

push.application.reliable.ports=100

Setup BlackBerry Simulator

To listen for incoming messages on BlackBerry, the Rhodes application will start when device powers on and will run a listener thread in the background. You will use the "push_port" option in the **rhoconfig.txt** to specify the listening port for incoming push messages. If "push_port" is not specified, default will be 100 as shown in the previous section.

> **For testing push, the BlackBerry simulator requires a 'kickstart' process. When the simulator starts, open the web browser on the simulator and navigate to any web page (i.e. http://m.google.com/). Now your simulator is ready to receive push messages.**

Now you are ready to **testing Push in the RhoConnect Web Console**.

Setup BlackBerry Device

To test push on a BlackBerry device, you will need to use a device that is configured with a BES server. Then, all you need to do is **build your Rhodes application for the device**.

BlackBerry 5.0 Push service

Rhodes 2.2 and higher support new **Blackberry Push Service**

Define in **rhoconfig.txt** which push to use (if empty or missed only MDS push is supported).

```
push_options = 'mds;push_service'
```

These parameters from **PushApplicationDescriptor**. Set them in **rhoconfig.txt**:

```
push_service_url  = 'https://pushapi.eval.blackberry.com/'
push_service_port = 20558
push_service_appname = 'RhoTest'
push_service_type = 'BPAS'
```

Alert Audio File Setup for Blackberry

In case of Blackberry, if the audio file is in the public folder, the file name will be `/apps/public/alerts/test-file.mp3`.

Push Callback

Rhodes applications can also handle PUSH notifications that didn't come from a RhoConnect application.

In this example, we will get the device ID so we can register the device with some push server:

```
System.get_property('device_id')
```

Application can set push callback to process any commands from server:

```
System.set_push_notification("/app/Settings/push_notify", '')
```

Callback parameters:

- since Rhodes 2.0.4 : contain push message parameters as hash
- rhodes < 2.0.4 : message – contain server push message body

Callback return:

- empty string – do not run Rhodes push command processing
- "rho_push" – to run rhodes push command processing (for alerts, sounds, etc…)

Example:

```
def push_notify
  puts 'push_notify: ' + @params.inspect
  "rho_push"
end
```

Push Payload

The Rhodes push payload allows more than one operation in a single message.

There are no required operations. There are default operations – if operation is not specified, no default operation will be performed.

Payload may include following operations which client will perform when it receives the PUSH message:

do_sync – do sync on spec specified sync source; use "all" to sync all sources

```
# one or more sources, or "all" can be specified
do_sync = "Product,Customer"
```

alert – bring app upfront and show specified message

```
alert = "Some message"
```

vibrate – vibrate for the specified number of milliseconds, up to 25500; if 0 or no duration is specified, it will vibrate for 2500 millisecond.

To enable vibrate in your rhodes application in build.yml:

```
capabilities:
 - vibrate
```

Example of vibrate call:

```
vibrate = 2000 #=> duration in milliseconds
```

sound – play specified file if media type supported by the phone. iPhone will ignore media-type

parameter.

```
sound = "hello.mp3"
```

File should be included to the application bundle in `/public/alerts` folder.

Media type should be either specified explicitly or may be recognized from file extension. Known file extensions are: .mp3 – audio/mpeg; .wav – audio/x-wav.

Alerts

In your controller, you may call on system alert methods to show popup, vibrate, or play audio file.

show_popup – bring app upfront and show specified message

```
Alert.show_popup "Some message"
```

You can also customize popup window's title, icon, buttons and specify callback to be called on button click:

```
Alert.show_popup( {
    :message => 'Some message',
    :title => 'Custom title',
    :icon => '/public/images/icon.png',
    :buttons => ["Yes", "No",
        {:id => 'cancel', :title => 'Cancel all'}],
    :callback => url_for(:action => :on_dismiss_popup) } )
```

Popup window always close after clicking on any button.

- :message – text to be displayed in popup window
- :title – title of the popup window
- :icon – image to be displayed in popup window. It's value can be one of predefined values or path to image file. Predefined values:
    - :alert – '!' icon
    - :question – '?' icon
    - :info – informational icon
- :buttons – array of buttons for popup window. Each button defined by its id and title. Button can be specified by Hash with :id and :title keys or just String – in this case both id and title will be set to this value.
- :callback – url to be called when any of the button will be clicked. This callback will be called with @params hash containing three keys: :button_id, :button_title and :button_index.

Example:

```
def on_dismiss_popup
  id = @params[:button_id]
  title = @params[:button_title]
  index = @params[:button_index]
  if id == 'Yes'
    # Handle 'Yes' button
  elsif id == 'No'
    # Handle 'No' button
  elsif id == 'cancel'
    # Handle 'Cancel all' button
  end
end
```

hide_popup – close current opened popup

```
```

```
Alert.hide_popup
```

vibrate – vibrate for the specified number of milliseconds, up to 25500; if 0 or no duration is specified, it will vibrate for 2500 millisecond.

```
Alert.vibrate(duration_in_milliseconds)
```

play_file – play specified file if media type supported by the phone. File should be included to the application. For example, if file is in public folder, file name will be `/apps/public/test-file.mp3` Media type should be either specified explicitly or may be recognized from file extension. Known file extensions are: .mp3 – audio/mpeg; .wav – audio/x-wav

```
Alert.play_file(file_name.ext, media_type)
```

show_status – show status messages

```
Alert.show_status(title, status_text, hide_button_label)
```

Status window will close after clicking on hide button.

- status_text – text to be displayed in status window
- hide_button_label – label of hide button

For example you can use Alert.show_status in sync notification callback:

```
def sync_notify
  status = @params['status'] ? @params['status'] : ""
  Alert.show_status(
    "Status",
    "#{@params['source_name']} : #{status}",
    Rho::RhoMessages.get_message('hide'))
end
```

Timer

In your controller, you may start timer using Rho::Timer class:

start – start timer and call callback after some interval

```
Rho::Timer.start(interval_milliseconds, callback_url, callback_data)
```

For example:

```
Rho::Timer.start(5000, (url_for :action => :timer_callback), "test")
```

stop – stop timer by callback

```
Rho::Timer.stop(callback_url)
```

Screen rotation

When user rotate device, Rhodes update view corresponding to new orientation. To get notification about rotation use callback:

```
System.set_screen_rotation_notification(callback, params)
```

Callback will be called when screen rotate (available since Rhodes 2.0). Callback parameters will be: width, height, degrees

Sample

See controller and view in the /app/ScreenRotation folder of the **System API Samples application** for more information.

Run external application

Use System.open_url : you can provide any url with any schema(http, file etc), depending of platform will be run associated application to handle this url

```
System.open_url('http://www.rhomobile.com')
```

Use rho_open_target=_blank in html link. Note that jQuery Mobile or other javascript library may disable this attribute.

```
<a href="http://www.google.com/?rho_open_target=_blank">Open Google in ex
```

Sample

See CustomUri of **system API sample application** for example.

Run rhodes application from browser

Android

On Android it is possible to start rhodes app from browser by http link or redirect response. It is also possible to register custom URI scheme for the app. At Android the URI must strictly follow standard URI rules. Additionally Android Chrome browser makes additional restriction: the host part of the URI must be resolvable. By default the http://rhomobile.com/com.vendor.appname URI format is used by Rhodes. You can add additional path and query parameters to the URI and handle it by application code. It is possible to customize the URI with build.yml settings. Default Rhodes URI contains four parts: – scheme – host name – path prefix – optional path and query You can customize sheme and host name parts with following build.yml parameters:

```
android:
    URIScheme: myapp
    URIHost: www.myhost.com
```

In case if URIHost is specified no path prefix will be used. If no custom URIHost then default one is used (rhomobile.com) and path prefix must contain java package name of your application. Optional path and query just passed to your application for further processing.

> **Google recommends to always use common scheme such as ‘http’, ‘https’, ‘ftp’, etc.**

http scheme example:

```
<a href="http://rhomobile.com/com.rhomobile.rhodessystemapisamples">Open
```

Custom scheme examples:

```
android:
    URIScheme: rho
```

```
:::html
<a href="rho://rhomobile.com/com.rhomobile.rhodessystemapisamples">Open S


:::ruby
android:
    URIScheme: rho-sas
    URIHost: rhomobile.com

:::html
<a href="rho-sas://rhomobile.com">Open System API Samples</a>
```

iPhone

On iOS devices it is possible to start rhodes app by link with special registered URI scheme from browser. You should register custom URI scheme for the app. Custom URI scheme is set up in build.yml then this scheme is used followed by any string.

Custom scheme example:

```
iphone:
    BundleURLScheme: myapp

:::html
<a href="myapp:string_with_params">Open MyApp application</a>
```

In application you can get start params by

```
System.get_start_params()
```

For previous example start params will be “string_with_params”.

For execute you rhodes based application from enother rhodes based application use next code:

```
System.open_url('myapp:string_with_params')
```

MapView

MapView class provides an embeddable map interface, similar to the one provided by the Maps application. The following code would go into your controller and the map appears on a whole page.

```
map_params = {
     :provider => 'Google',
     :settings => {:map_type => "hybrid",:region => [@params['latitude'],
                   :zoom_enabled => true,:scroll_enabled => true,:shows_u
                   :api_key => 'Google Maps API Key'},
     :annotations => [{:latitude => @params['latitude'], :longitude => @p
                       {:street_address => "Cupertino, CA 95014", :title =
                        :url => "/app/GeoLocation/show?city=Cupertino"},
                       {:street_address => "Santa Clara, CA 95051", :title
                        :url => "/app/GeoLocation/show?city=Santa%20Clara"
}
MapView.create map_params
```

Provider:

There are two providers supported now – ‘Google’, ‘ESRI’, “RhoGoogle” and “OSM”.

- ‘Google’ supported on iPhone, Android and BB.

- ‘ESRI’ supported on iPhone, Android, BB and WM.
- ‘RhoGoogle’ own GoogleMap provider based on Google Static Map API. Have additional features (offline caching for example). Supported on Android and WM
- ‘OSM’ – Open Street Map, support only ‘roadmap’ map type. Supported on Android and WM

For use ESRI maps in your projects on iPhone:

- Install **ESRI iOS SDK** – use default folders during installations !
- Add “esri” to your applications extensions list in build.yml file.

To use native Google map view on Android:

- Install Google Add-on API
- Get **Google Maps API key**

- Add ‘mapping’ and ‘apikey’ parameters to your build.yml

```
android:
    mapping: yes
    apikey: <YOUR-API-KEY>
```

- Add ‘network_state’ to capabilities list in your build.yml.

```
capabilities:
    - network_state
```

Map settings:

- map_type – widget may display maps of three types: standard, satellite, and hybrid
- region – two types allowed:
    - [latitude,longitude,latitudeDelta,longitudeDelta]. The area currently displayed by the map view. The region encompasses both the latitude and longitude point on which the map is centered and the span of coordinates to display. The span values provide an implicit zoom value for the map. The larger the displayed area, the lower the amount of zoom. Similarly, the smaller the displayed area, the greater the amount of zoom.
        - latitude,longitude – map coordinate of the region center
        - latitudeDelta – the amount of north-to-south distance (measured in degrees) to display on the map. Unlike longitudinal distances, which vary based on the latitude, one degree of latitude is always approximately 111 kilometers (69 miles).
        - longitudeDelta – the amount of east-to-west distance (measured in degrees) to display for the map region. The number of kilometers spanned by a longitude range varies based on the current latitude. For example, one degree of longitude spans a distance of approximately 111 kilometers (69 milies) at the equator but shrinks to 0 kilometers at the poles.
    - {:center => center, :radius => radius}. Another way to define area displayed by the map view.
        - center – string describing center of area. Can be any string allowed by google geocoding service. Examples: “1 infinite loop, cupertino, ca 95014”, “California, USA”, “37.365519,-121.982918”
        - radius – radius of displayed area measured in degrees
- zoom_enabled – true if zoom of the map is enabled
- scroll_enabled – true if scrool of the map is enabled
- shows_user_location – true if current user location is displayed on the map
- api_key – Maps JavaScript API Key (**sign up for it here**). Note that this parameter is ignored on Android! For Android you should get another key and specify it in the app’s build.yml as described **here**

Annotations – array of map annotation objects (list of pins on the map). Annotation:

- latitude,longitude – map coordinate of the annotation
- street_address – if map coordinate is not specified, framework will attempt to obtain it using provided street address from google geo-coding service

- title – title of the annotation callout
- subtitle – subtitle of the annotation callout
- url – url to follow when user click on the callout button
- image – image file name
- image_x_offset – int horizontal coordinate of image anchor (ancor point placed to annotation point on map) . Set to image width/2 for center
- image_y_offset – int vertical coordinate of image anchor (ancor point placed to annotation point on map). Set image height/2 for center
- pass_location – then true, location coordinates added to url in the format: latitude=xx.xxx&longitude=xx.xxx

Map settings in rhoconfig.txt :

- ESRI_map_url_roadmap – URL of ESRI roadmap tile map server (example: 'http://server.arcgisonline.com/ArcGIS/rest/services/World_Street_Map/')
- ESRI_map_url_satellite – URL of ESRI satellite tile map server (example: 'http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/')
- OSM_map_url_roadmap – URL of OSM tile map server (example: 'http://tile.openstreetmap.org/')

Enable file caching for map tiles – file cache can use for offline map browsing:

```
MapView.set_file_caching_enable(1)
```

Preload map tiles for region (0<=zoom<=18):

```
def preload_callback
     puts '@@@@@@@@@@      Preload Callback       STATUS['+@params['stat
end

def preload_map
    options = { :engine => 'OSM',
        :map_type => 'roadmap',
        :top_latitude => 60.1,
        :left_longitude => 30.0,
        :bottom_latitude => 59.7,
        :right_longitude => 30.6,
        :min_zoom => 9,
        :max_zoom => 11
      }
    total_tiles_for_preload_count = MapView.preload_map_tiles(options,
    redirect :action => :index
end
```

Sample

See GeoLocation/controller.rb of **system API sample application** for some of the examples of how to use MapView class.

File system access

Rhodes client file system structure

```
<rhodes root>  #system-dependent path
  apps         #Rho::RhoApplication::get_base_app_path
    app        #Rho::RhoApplication::get_app_path('app') - contain models
      model1   #Rho::RhoApplication::get_model_path('app','model1')
  public       #contains files from application public folder
  db           #contains schema and data files
    db-files   #contains files stored in database(blobs)
               # for file paths from camera callback etc: Rho::RhoApplica
               # to create file path for blob: Rho::RhoApplication::get_b
  lib          #contains rho framework library files. Blackberry does not
```

```
    RhoLog.txt   #application log
```

Read\write file example

```
fileName = File.join(Rho::RhoApplication::get_base_app_path(), 'myfile.tx
File.open(fileName).each do |line|
end

fileNameW = File.join(Rho::RhoApplication::get_base_app_path(), 'tempfile
f = File.new(fileNameW)
f.write('test')
f.close

#binary file read\write
#to read file from blob attribute use Rho::RhoApplication::get_blob_path(

file_testname = File.join(Rho::RhoApplication::get_model_path('app','Data
test_content = File.binread(file_testname)

file_name = File.join(Rho::RhoApplication::get_base_app_path(), 'temp.png
f = File.new(file_name, "wb")
f.write(test_content)
f.close
```

Platform notes

Blackberry

Only read from files are supported.

Simulator files folder (4.6 and bigger) – `<sdk root>/components/simulator/sdcard/rho/<appname>`

Device files folder can be found using Media/Explore.

iPhone

Simulator files folder – run search for RhoLog.txt from the drive root. Files are placed inside simulator folder.

Application can creates folders and files under apps and db roots.

Rhodes client file system structure on iOS platform

```
<rhodes root>  # /Library/Caches/Private Documents/
  apps         #Rho::RhoApplication::get_base_app_path
    app        #Rho::RhoApplication::get_app_path('app') - contain models
      model1   #Rho::RhoApplication::get_model_path('app','model1')
    public       #contains files from application public folder
  lib          #contains rho framework library files. Blackberry does not
  db           #contains schema files
  RhoLog.txt   #application log
<rhodes user root>  # /Documents/
                    apps     # user path   Rho::RhoApplication::get_user_
  db           #contains data files
    db-files   #contains files stored in database(blobs)
               # for file paths from camera callback etc: Rho::RhoApplica
               # to create file path for blob: Rho::RhoApplication::get_b
```

On iOS platform only files stored in /Documents/ bakuped in iCloud etc. Before Rhodes 3.3.2 all files stored in /Documents/ but Apple require do not placing files in Documents folder except user produced files. Now all files except databases stored in /Library/Caches/Private Documents/ – files

in this folder do not bakup. If you want make any files should be bakup – use “user folder” for it – Rho::RhoApplication::get_user_path. All you files created in old version in Rho::RhoApplication::get_base_app_path now should be open in Rho::RhoApplication::get_user_path.

Also there are few additional parameters in build.yml (use it only if you want tune iOS specific folder scheme for some reason).

- iphone_db_in_approot – if 1 then place databases files into
- iphone_set_approot – set to one of the three folders (property value – folder): “Documents” – /Documents/, “Library_Caches” – /Library/Caches/Private Documents/, “Library_Private_Documents” – /Library/Private Documents/
- iphone_userpath_in_approot – is 1 then will be in the same place with

change low level parameters in build.yml example

```
iphone_db_in_approot: 1
iphone_set_approot: library_Private_Documents
iphone_userpath_in_approot: 1
```

write file in user folder example

```
fileNameW = File.join(Rho::RhoApplication::get_user_path(), 'tempfile.txt
f = File.new(fileNameW, 'w+')
f.write('my own file !')
f.close
```

Also you can set special iOS attiribute “do not bakup” to any files or folders or files related to database :

setup “do not bakup” attribute for files related database conaint Model ‘Product’

```
require_model 'Product'
db = Rho::RHO::get_src_db('Product')
db.set_do_not_bakup_attribute(1)
```

Windows Mobile

Device/simulator files folder root – Program Files//rho

Desktop win32 simulator

To use client with the HTTP proxy you should pass its URL as the first command line argument `–http_proxy_url=http://<login>:<passwod>@<host>:<port>` or add it to rhoconfig.txt.

Shutdown hook

Like any application written in Ruby, rhodes can register a shutdown hook. Shutdown hook is routine registered for execution when the program exits. It usefull for cleanup on exit, saving program states, etc. To create you own shutdown hook you should add at_exit block to the file application.rb. For example:

```
at_exit do
  #delete all temporary files
  ...
  ...
end
```

Media Support

Rhodes can play video/audio files in native Internet Browser. So application developer can just add link to online or local audio/video file:

```
<a href="http://videos3.netprofesseur.com/1/videos/130/BRICOLAGE_Changer_
```

Application can download file to file system using **AsyncHttp.download_file** and than put link **to this file** to view.

Sample

See app\Media of **System API Sample application** as an example.

Using Hyperlinks for Email, Phone Dialing, SMS and others

You can allow your users to send email messages, call phone numbers and send SMS messages using the hyperlink (`<a href="...">`) syntax. Please note not all of these examples could work on simulators! Use real devices for test. Examples are shown below.

To make phone calls enable the phone capability. This is done by adding the following lines to build.yml:

```
capabilities:
  - phone
```

mailto

```
<a href="mailto:test@host.com?subject=testing123">Mailto</a>
```

Note, even for an empty address, you must add the @ symbol: **Mailto**

Blackberry: if you need cc, bcc feilds, use rhomailto scheme:

```
<a href="rhomailto:test@host.com?cc=geny@mail.ru&bcc=vasy@gmail.com&body=
```

tel

```
<a href="tel:1-555-531-3255!8335033#!#!9582#">Tel</a>
<a href="wtai://wp/mc;5195551212" title="Call">Work Tel</a>
<a href="wtai://wp/mc;5195551213" title="Call">Home Tel</a>
```

Note, the WML tel description can be found **here**.

sms:

```
<a href="sms:+3581234567">Send SMS to us </a>
```

Open link in external application (browser for http:// links):

```
<a href="http://www.google.com/?rho_open_target=_blank">Open Google in ex
```

Open appstore on iphone: **http://wiki.akosma.com/IPhone_URL_Schemes#App_Store**

jQuery/jQuery Mobile

When using jQuery and/or jQuery Mobile in application code, you cannot use usual html links for hyperlinks. You need to call controller action and call WebView.navigate from there:

```
#Ajax call of controller method:
$.get("/app/TestController/send_mail",function(data) {});

#TestController method:
def send_mail
    WebView.navigate( 'mailto:test@host.com' )
    #to open url inexternal application you can use System.open_url
end
```

Blackberry

On Blackberry 5.0 and later you can use JavaScript **blackberry.launch object** to create SMS, make Phone call etc.

BlackBerry network connection

Rhodes may use any Blackberry network connection available on the device. What particular network connection will be selected depends on the url suffix used to connect. Here is the algorithm:

1. Rhodes enumerate device Service Books at application start to identify Wifi, BES (has priority over BIS-B) and BIS-B transport:
    - if WIFI exist we set wifi_postfix = ";interface=wifi"
    - if BES exist we set url_postfix = ""
    - If BIS-B exist and NO BES: url_postfix = ";deviceside=false;ConnectionType=mds-public"
    - If no BES or BES exist or in simulator mode : url_postfix = ";deviceside=true"
2. When application make network call, Rhodes create network connection:
    - if wifi_postfix is not empty and WIFI network available: add wifi_postfix to url
    - add url_postfix
    - make network connection
3. If connection creation failed, Rhodes try to connect without wifi_postfix(if exists) and then without url_postfix (if exists)
4. There are several **rhoconfig.txt** parameters which can modify this process (mostly for the testing purposes):

Do not use `;deviceside=true` suffix:

```
no_deviceside_postfix = 0
```

Set `url_postfix` to specific value, without enumerating Service Books records

```
bb_connection_postfix = ''
```

BlackBerry Browser Support

Capabilities of BlackBerry WebView control differ quite significantly form other OS-es as well as from one version of BB OS to another. And even with the single BB OS you may have different capabilities depending on what WebView do you use (see full browser capability described below).

Check out the **BlackBerry Browser Version 4.2 Content Developer Guide** to see what HTML, CSS and JavaScript supported on BlackBerry.

One of the limitation is that simple submit element in the form does not work (this issue was fixed in

Rhodes > 2.2.4):

```
<form id="user_edit_form"
   method="POST"
   action="<%=url_for(:action => 'do_login')%>" selected="true">
<input type="submit" value="Login"/>
</form>
```

Solution:

```
<form id="user_edit_form"
   method="POST"
   action="<%=url_for(:action => 'do_login')%>" selected="true">
</form>
<a href="#" onclick="document.forms[0].submit(); return false;">
    <img src="/public/images/button_next.png" />
</a>
```

Blackberry Touch screen (4.7, 5.x)

There are some issues Rhodes developers encountered developing for Blackberry Touch screen 4.7 and 5.x.

Links with aligned images are not clickable:

```
<a href="/app/WelcomeScreen/do_info">
    <img src="/public/images/bb_info_btn.png" align="left" />
</a>
```

Solution: remove alignment or add text to link

Links with div inside issue:

```
<a href="/app/Login"><div style="height:10px"></div>Login</a>
```

Solution: move dive out of `<a>`:

```
<div style="height:10px"></div><a href="/app/Login">Login</a>
```

Links with style display:block:

```
<a href="/app/Login"
    style="display:block;background: url('test.png') no-repeat 97% 50%;">
    Login
</a>
<a href="/app/Login"
    style="display:block;border-Top: 1px solid #bbb;">
    Login
</a>
```

Solution: remove display:block from a element

Blackberry 5.0 and above full browser

Blackberry 5.0 and above has new BrowserField class, which support javascript, ajax and extended css. To use it set in **rhoconfig.txt**:

```
# use full browser only on BB 5.0 and above
```

```
use_bb_full_browser=5

# use full browser only on touch devices including 5.0
use_bb_full_browser=touch
```

WebView.execute_js is also supported in this mode.

> on Blackberry 6.0 and above it is recommended to use full browser mode. Because overwise some UI elements like combobox is not selectable by trackball:

```
# use full browser only on BB 6.0 and above
use_bb_full_browser=6
```

Blackberry full browser

Available for Blackberry 4.6 and above

Support AJAX and better support of CSS

On non-touch screen looks like usual browser app, so not very useful because it uses pointer cursor. On Touch screen devices no visual difference between full browser and browser field (default mode).

To enable on all devices – add to **rhoconfig.txt**:

```
use_bb_full_browser=1
```

To enable on Touch screen devices – add to **rhoconfig.txt**:

```
use_bb_full_browser='touch'
```

Submit form issue:

```
<head>
<script type="text/javascript">
function submitform()
{
    document.forms["login"].submit();
}
</script>
</head>
<form id="user_edit_form"
   method="POST"
   action="<%=url_for(:action => 'do_login')%>" selected="true">
<a href="javascript: submitform()">Login</a>
</form>
```

Solution:

```
<form id="user_edit_form"
   method="POST"
   action="<%=url_for(:action => 'do_login')%>" selected="true">
    <a href="#" onclick="document.forms[0].submit(); return false;">
        <img src="/public/images/bb_logon_btn1.png" />
    </a>
</form>
```