Locly App Card API

App Version: 6.3 **Release Date:** 9th December 2014

Introduction

The Locly App Card API allows cards viewed within the Locly App to take advantage of extra functionality provided by the app itself.

When cards are rendered the contents are downloaded from the Locly webservers and cached within the App. Cards and card contents are automatically updated every 10-15 seconds by the app when the publisher updates the card contents.

Getting started

To create a custom Locly card that runs 3rd party code you will first need to create a HTML card on the Locly CMS. Within the HTML card type you are able to upload any number of files with any format. You must upload an index HTML file with the name suffix for the language you are targeting. For example this could be `index_en.html` if you are targeting English.

The HTML file can reference any other files within its own card in the same way a normal webpage would by using a relative URL. This for example could consist of:

```
<link rel="stylesheet" type="text/css" href="mystyle.css">
<script type="text/javascript" src="script.js"></script>
<img src="image.png" />
```

Caching

Any file uploaded to the Locly CMS and referenced from the HTML card will automatically be cached and made available for offline use. If you are using resources externally from the Locly eco-system these will not be automatically cached by the app.

iOS handles Audio and Video tags differently to most other resources and because of this, these are not automatically cached by the app. To provide caching for these items local card assets are served from an internal web-server that is only available to the running card. The JavaScript API provides methods to robustly update Audio and Video tags which is documented further on within this document.

Localstorage

If you need to use HTML5 LocalStorage within a card you should be aware that each card is hosted on its own subdomain to prevent conflicts. If you require storage that is accessible by multiple cards you should use the Project Storage API provided by the Card JavaScript API (see below) and manage any

conflicts gracefully.

JavScript Debugging

The Locly App provides a JavaScript module that can help with Debugging hand coded cards. This module consists of two parts, firstly a script that runs within the card and secondly a server that runs on a local network. It is **not** recommended that you leave the debug library in a card when publishing cards for production as the card will continue to post log messages on the local network.

To run the server you will need python installed on a machine on the local network. You can find the lcdb.py server in the Locly SDK and then from the command line you can run

```
python lcdb.py
```

The server will now start listening on port *8000* for incoming log messages from cards.

In the card you will need to include a JavaScript File along with the location of your debug server. Failing to set the *data-target* attribute of your local server will result in your log messages not being received.

```
<head>
    <script type="text/javascript" src="/LoclyClasses/LoclyCardDebugBridge.js"
data-target="192.168.0.10:8000"></script>
</head>
```

When the card initializes it will automatically post a log message to your server to inform it that it is running. Now from anywhere in your card you can access the following methods and provide any type of variable to be logged...

```
lcdb.log('A log');
lcdb.log('A log with multiple objects', true, {}, []);
lcdb.warn('A warning');
lcdb.warn('A warning with some more info', 'More info', {}, [], true, 10);
lcdb.error('An Error');
lcdb.error('An Error with an object', {error:true});
```

JavaScript API

The Locly App provides a JavaScript API that wraps many common tasks that a card may wish to perform. To access this API you need to include it in the head of your card. You can do this by referencing the following script:

```
<head>
    <script type="text/javascript" src="LoclyClasses/LoclyCard.js"></script>
</head>
```

At run time this script will be evaluated and included in the card. The API sits under the `window.locly` namespace to avoid naming conflicts.

When making calls to the card API you should be aware of your current permission space. Some JavaScript methods allow a card to make changes to the Project, Place and also other Cards. The scope of these changes however is always limited to the scope of the current project.

JavaScript API : Visibility

You are able to write the visibility of Places and Cards from the API using methods such as `locly.card.setVisibility`. These methods accept visibility parameters which are a set of conditions that are evaluated before an item is presented to the user. If **any one** of the visibility conditions types evaluates to **false** the item is **not** shown. If you are setting the visibility of any items you should use the following as reference before submitting the visibility fields. Documentation on setting the visibility of individual items follows later on in this document.

When submitting a visibility clause you can either submit an object `{}` or a set of objects in an array `[{}, {}, ... ]`. Each visibility clause must have a `type` field and may have other fields to define the behaviour. Below are the available visibility clauses

shown

Always show the item. `{ type : 'shown' }`

hidden

Always hide the item. `{ type : 'hidden' }`

default

Execute the default behaviour. For cards this evaluates a shown clause. For places the place is only shown in *nearby* when the device is nearby to the beacon and is always shown in *all*.
`{ type : 'default' }`

nearby_strict

Only ever show the item when nearby to the assigned beacon. (This omits it from the all list when not nearby). `{ type : 'nearby_strict' }`

time

Only show the item when the defined time is true. Days should be a 7 character string, using any character to indicate a show condition but an underscore `_` to indicate a hide condition. The string starts on Sunday and ends on Saturday. Time should be an array of two items with the first item defining the time the item should be shown and the second item defining the time the item should be hidden.
`{ type : 'time', days : 'SMTWTFS', time : ['11:59', '15:00']}` *Time visibility is refreshed periodically and may not take effect for up to a minute.*

RSSI

Only show the item when the RSSI of its defined beacon is between bounds. RSSI is measured in negative values meaning that a high value such as -1 means you're extremely close and a low value such as -999 means you're really far away. Omitting `closerThan` defaults it to `-999` and omitting `furtherThan` default it to `-1`.
`{ type : 'RSSI', 'closerThan' : -999, 'furtherThan' : -1 }` *RSSI visibility is currently only available on a place. Setting this on a card will result in the card being hidden.*

Simple Example

A simple visibility condition could be considered one that always shows a card. You could for example execute the following code

```
window.locly.card.setVisibility(locly.card.currentId(), { type : 'shown' });
```

Advanced Example

A more complex example, could be that you want a place to only appear nearby, in the morning and evening. To achieve this you could execute the following

```
window.locly.place.setVisibility(locly.place.currentId(), [
    {
        type : 'nearby_strict'
    },
    {
        type : 'time',
        days : 'SMTWTFS',
        time : ['00:00', '11:59']
    },
    {
        type : 'time',
        days : 'SMTWTFS',
        time : ['18:00', '23:59']
    }
]);
```

full documentation for the place and card follows later on in this document...

JavaScript API : Project

```
locly.project.currentId = function()
```

- **Return [string]** the id of the current project

JavaScript API : project.store

A key value database that persists across all cards within a project. Keys stored are not resolved against card ids, thus all cards within the project should either agree a common naming convention to avoid conflicts or deal with conflicts gracefully.

```
locly.project.store.get = function(key)
```

- **key [string]** the key of the value to get
- **Return [object, array, string, number, null, undefined]** the original object stored or undefined if no object has been stored against the given key

```
locly.project.store.set = function(key, value)
```

- **key [string]** the key to set the value against
- **value [object, array, string, number, null, undefined]** the value to set in the database. Note that values are available in memory immediately but only written to disk every 250ms for efficiency reasons.

```
locly.project.store.keys = function()
```

- **Return [array]** a list of currently in the database

JavaScript API : Place

```
locly.place.currentId = function()
```

- **Return [string]** the id of the current place

```
locly.place.setVisibility = function(placeId, flag, callback, error)
```

- **placeId [string]** the id of the place to set the visibility
- **flag [Visibility]** the condition to set on the place
- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

```
locly.place.setMultipleVisibility = function(flags, callback, error)
```

Sets the visibility of a set of places overwriting the pre-set visibility provided by the Locly CMS.

- **flags [object]** an object mapping place ids to their visibility conditions to set more than one visibility condition at once
- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

```
locly.place.visibleAnywhere = function(placeId, callback, error)
```

Queries the current visibility of a place

- **placeId [string]** the id of the place to check
- **callback [function]** executed on completion. If the place is visible in any context for the current user the function is provided with *true*, otherwise *false*.
- **error [function, optional]** executed on error and provided with the error

```
locly.place.visibleNearby = function(placeId, callback, error)
```

Queries the current visibility of a place in the nearby context

- **placeId [string]** the id of the place to check
- **callback [function]** executed on completion. If the place is visible in the nearby filter for the current user the function is provided with *true*, otherwise *false*
- **error [function, optional]** executed on error and provided with the error

JavaScript API : Card

```
locly.card.currentId = function()
```

- **Return [string]** the id of the current card

```
locly.card.setVisibility = function(cardId, flag, callback, error)
```

- **cardId [string]** the id of the card to set the visibility
- **flag [Visibility]** the condition to set on the card
- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

```
locly.card.setMultipleVisibility = function(flags, callback, error)
```

Sets the visibility of a set of cards overwriting the pre-set visibility provided by the Locly CMS.

- **flags [object]** an object mapping card ids to their visibility conditions to set more than one visibility condition at once
- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

```
locly.card.visibleAnywhere = function(cardId, callback, error)
```

Queries the current visibility of a card

- **cardId [string]** the id of the card to check
- **callback [function]** executed on completion. If the card is visible in any context for the current user the function is provided with *true*, otherwise *false*.
- **error [function, optional]** executed on error and provided with the error

```
locly.card.visibleNearby = function(cardId, callback, error)
```

Queries the current visibility of a card in the nearby context

- **cardId [string]** the id of the card to check
- **callback [function]** executed on completion. If the card is visible in the nearby filter for the current user the function is provided with *true*, otherwise *false*
- **error [function, optional]** executed on error and provided with the error

```
locly.card.open = function(cardId, callback, error)
```

Opens the card within the Locly app. Before the card is opened its visibility will be checked to ensure it can be opened. The card can be within a different project.

- **cardId [string]** the id of the card to open
- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

JavaScript API : Beacon

This provides a way for a card to transmit a beacon and listen on transmitting beacons. A card can only transmit one beacon at a time and once the card is closed the beacon will stop transmitting. The app only listens for a subset of beacon UUID's as per the restrictions provided by iOS.

```
locly.beacon.startTransmitting = function(uuid, major, minor, callback, error)
```

Transmits a beacon with the given credentials from this devices.

- **uuid [string]** the UUID of the beacon to transmit. This can optionally contain dashes.
- **major [int]** the major number of the beacon to transmit with a minimum value of 0 and maximum of 65025.
- **minor [int]** the minor number of the beacon to transmit with a minimum value of 0 and maximum of 65025.
- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

```
locly.beacon.stopTransmitting(callback, error)
```

Stops transmitting a beacon from this device

- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

```
locly.beacon.currentlyTransmitting(callback, error)
```

Provides information on the currently transmitting beacon

- **callback [function]** executed on completion and provided with the following
    - **isTransmitting [bool]** true if the device is transmitting, false otherwise
    - **uuid [string]** the UUID of the beacon that is being transmitted
    - **major [int]** the major number of the beacon
    - **minor [int]** the minor number of the beacon
- **error [function, optional]** executed on error and provided with the error

```
locly.beacon.visible(callback, error)
```

Gets the list of beacons the device can currently see and the information about those beacons

- **callback [function]** executed on completion and provided with an array of visible beacons. Each item in the array is an object containing the following keys
    - **type [string]** the type of beacon, for example `qr` for a qr-code beacon, or `rf` for a radio frequency beacon
    - **id [string]** the unique identifier for this beacon made from a composite of UUID, major and minor
    - **uuid [string]** the UUID of the beacon
    - **major [int]** the major number of the beacon
    - **minor [int]** the minor number of the beacon
    - **rssi [int]** the RSSI value of the beacon
    - **averagedRSSI [int]** the RSSI value of the beacon averaged over a short period of time
    - **distance [float]** the approximate distance from the beacon in meters

    - **averagedDistance [float]** the approximate distance from the beacon averaged over a short period of time
    - **proximityString [string]** a string of `immediate`, `near`, `far` or `unknown` dependent on the beacons distance.

- **error [function, optional]** executed on error and provided with the error

JavaScript API : project.poll

The beacon poll API provides a way for a group of nearby devices to participate in a poll even without internet connectivity. It does this by manipulating the namespace provided by a beacons major and minor number and transmitting on that beacon. You can not transmit a poll result and beacon simultaneously.

```
locly.beacon.poll.transmitResult = function(index, callback, error)
```

Transmits a users choice in a poll using an iBeacon.

- **index [int]** the index of the result to transmit with a minimum of 0 and maximum of 15.
- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

```
locly.beacon.poll.stopTransmitting = function(callback, error)
```

Stops transmitting the users choice in a poll.

- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

```
locly.beacon.poll.results = function(callback, error)
```

Gets the currently visible set of poll beacons and extrapolates the votes that each device has cast. This is not persistent, in that once each device has stopped transmitting the results will disappear.

- **callback [function]** executed on completion and provided with an array of length 16. Each item in the array is an integer which indicates the amount of visibile iBeacons voting for each index. Note that the current devices chosen index is not included in these counts.
- **error [function, optional]** executed on error and provided with the error

JavaScript API : Treasurehunt

Places can be part of a treasure hunt. Normally the following place is unlocked automatically by entering any card within a place. This can be customised from the Locly CMS and the card can add extra tasks for example, before unlocking the next location

```
locly.treasurehunt.isPlacePartOfTreasurehunt = function(placeId, callback, erro
r)
```

Provides a method of checking if a place is part of a treasure hunt

- **placeId [string]** the id of the place to check
- **callback [function]** executed on completion and provided with *true* if this place is part of a treasure hunt, *false* otherwise
- **error [function, optional]** executed on error and provided with the error

```
locly.treasurehunt.isPlaceLocked = function(placeId, callback, error)
```

Provides a method of checking if a place is still locked in a treasure hunt. A locked place is one that the user will not see until it has been unlocked.

- **placeId [string]** the id of the place to check
- **callback [function]** executed on completion and provided with *true* if this place is still locked in a treasure hunt, *false* otherwise.
- **error [function, optional]** executed on error and provided with the error

```
locly.treasurehunt.unlockPlace = function(placeId, afterSeconds, callback, erro
r)
```

Unlocks a place so that a user can find it. The place is still subject to its visibility conditions hence for example if it is set as hidden the user will still be unable to find it. You can set the amount of seconds that must pass before the user can find the place. This can be useful if you want to stop users completing the treasure hunt too quickly.

- **placeId [string]** the id of the place to set
- **afterSeconds [int]** the number of seconds to unlock this place after. To unlock immediately set 0
- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

```
locly.treasurehunt.relockPlace = function(placeId, callback, error)
```

Re-locks a place so that the user can no longer find it until it is unlocked again.

- **placeId [string]** the id of the place to lock
- **callback [function, optional]** executed on completion
- **error [function, optional]** executed on error and provided with the error

JavaScript API : Apps

```
locly.openUrl = function(url, callback, error)
```

Opens a url on the user device. The url can be a custom url scheme for another app causing the third party app to open on the users device.

- **url [string]** the url to open
- **callback [function, optional]** executed on completion and provided with *true* if the url could be opened on the device or *false* otherwise. *false* may be provided if for example the custom url scheme was for an app the device does not have installed
- **error [function, optional]** executed on error and provided with the error

JavaScript API : Utilities

```
locly.appVersion = function()
```

- **Return [string]** the version of the app currently running

```
locly.appName = function()
```

- **Return [string]** the name of the app. This will be different for customised apps provided to 3rd party publishers

```
locly.appIdentifier = function()
```

- **Return [string]** a culmination of the app version and app name providing a uniquely identifiable string for the running app.

```
locly.localUrlForAsset = function(url)
```

Translates a Locly url into the url for the asset on the local card server

- **url [string]** the Locly url to translate
- **Return [string]** the url for the asset on the local card server

```
locly.setLocalUrlForAudioOrVideo = function(element)
```

Updates an Audio or Video DOM element with the local card server url for its provided source

- **element [DOM element]** the DOM element to update

```
locly.setLocalUrlForAllMediaInDocument = function()
```

Updates all Audio and Video DOM elements in the current document to also have the local card server url for their provided sources

URL Schemes

The Locly App provides a number of privileged url schemes that a card can address to trigger special actions. These actions could, for example launch a widget or full screen image. You can trigger these from a user action or by evaluating JavaScript. Below for example are two ways you could open urls in safari

```
<a href="web://www.locly.com/">Click Me</a>
```

```
<script type="text/javascript">
    window.location.href = 'web://www.locly.com/';
</script>
```

Please note that before executing privileged url schemes you should ensure the document has loaded. If using anchor tags waiting for user to action the tag will ensure this, or if using JavaScript placing the script at the end of the document or in a `setTimeout` will also ensure this.

web://

Open a http url in safari. `http://locly.com` would become `web://locly.com`

webs://

Open a https url in safari. `https://locly.com` would become `webs://locly.com`

widget://

Opens a fullscreen widget. If you uploaded a widget in a zip file to the Locly CMS and it was named `mywidget.wdgt.zip` the url you would attempt to open would be `widget://mywidget.wdgt.zip`.

open://

Opens a file using the iOS file open protocol and picker. This could for example be to open a pages file in the page app. If you uploaded a pages files the Locly CMS and it was named `myfile.pages` the url you would attempt to open would be `open://myfile.pages`.

preload://

Preloads an asset in the background. This is useful sometimes if you want to load a widget or large file without requiring the user to wait. Twinned with the `open://` example above you could attempt to preload the page file by directing the card to `preload://myfile.pages`.

fsimg://

Opens an image fullscreen allowing the user to zoom in and out inspecting the image closer. If you uploaded an image file to the Locly CMS and it was named `bigimage.png` the url you would attempt to open would be `fsimg://bigimage.png`.

loclyapi://

Submits requests to the locly information API. **You should no longer use** this but instead use the JavaScript API which wraps the complexity of request serialisation and deserialsation for you.