RAPLETS

APPLICATIONS FOR THE RAPPORTIVE INBOX PLATFORM

This is version 0.3 (May 2010) of the Raplet documentation. For changes since previous versions, see the changelog in the appendix. For updates, please subscribe to the raplet-announce mailing list (http://groups.google.com/group/raplet-announce). For help, questions or feedback, please email the raplet-dev mailing list (http://groups.google.com/group/raplet-dev).

OVERVIEW

The aim of Rapportive is to give email users access to relevant contextual information as part of their normal email workflow. In its current form, Rapportive displays a sidebar next to an email thread in Gmail, showing information about one particular person. The information is collated from social networking sites and other sources. Rapportive chooses one sender or recipient of an email thread to show by default; if the user wishes to see information about another person, they can hover their mouse over that person's email address.

Soon after Rapportive was released, users started asking us whether we could add various other pieces of information to the sidebar. In response, we introduced Raplets, which are a flexible way for any web developer to insert content into Rapportive, and to provide rich integrations with external services.

A Raplet is a piece of code that runs on your server and as accessible via a URL. Any user can add Raplets to their sidebar by visiting raplets.com and entering the Raplet URL. We showcase a number of existing Raplets at raplets.com for the benefit of all Rapportive users; if you would like your Raplet to be added to the site, please contact support@rapportive.com.

Rapportive is currently implemented as a browser extension that dynamically modifies the Gmail page to insert the sidebar. However, Raplets are designed with future portability in mind. As Rapportive expands into other email clients and communication platforms, we will keep Raplets compatible with this specification, provided that it is technically feasible to do so.

USE CASES

Who would benefit from developing a Raplet? There are many potential use cases:

- Do you have a SaaS web application? You can offer additional value to your customers, drive loyalty and even reach new customers by integrating your service into their normal email workflow.
- Do you target consumers? With a Raplet you can drive engagement with your service directly inside email. People spend hours a day in their email; it's a valuable place to have a presence.
- As a business, you probably keep data about people in various internal databases. You can make a Raplet to present internal data, e.g. a customer's recent interactions with your product. Data can be on your local network; because it is accessed directly from your browser, it doesn't need to be on the public web.

HOW RAPLETS WORK

Let's run through an example to get you started quickly. You can also find code samples below.

YOUR SERVER AND URL

Your Raplet runs on your own server. You may use any programming language, and any hosting provider (or even a server on your local network or your local machine, provided that your browser can access it by HTTP). You can use cookies to identify users, if necessary (it's easiest if you authenticate users in a separate window, and set the cookie there). If your data is publicly accessible, it's probably fine for your server to be on the internet and not require authentication.

In this example, let's say you are developing the Raplet on your local machine, and you are serving it at the following URL: `http://localhost:8080/raplet.php`

Any valid URL is fine, with the following restrictions:

- The protocol must be `http` or `https`.
- The URL must not contain the question mark (`?`) or any query parameters. (We will append query parameters to it later.)
- The URL must not contain an anchor fragment (`#something` at the end).
- If you have a multi-user system, all users should be able to access your data via the same URL; you should distinguish users through cookies or query parameters (see the section on configuration below). This is particularly relevant if your application uses subdomains like `http://customername.example.com`; you will not be able to configure the subdomain individually for each customer.

REQUEST FORMAT

Raplets are implemented using JSONP — but don't worry if you've not heard about it before; it's easy.

Say you get an email from Rahul Vohra, whose email address is rahul@rapportive.com and whose Twitter username is @rahulvohra. If you have Rapportive and your Raplet installed, Rapportive will ask your browser to make a `GET` request to your Raplet URL, and include some information about Rahul in the query parameters. At the moment, the following parameters are included (URL-encoded as required for URLs):

- The `email` parameter contains the contact's email address, in this case `rahul@rapportive.com`. It may contain a mixture of uppercase and lowercase letters; be sure to make your search case insensitive.
- The `name` parameter contains the contact's full display name, in this case `Rahul Vohra`. Don't forget that it may contain arbitrary Unicode characters; we encode them as UTF-8.
- The `twitter_username` parameter contains the contact's Twitter username without @ prefix, in this case `rahulvohra`. Like the email address, it may contain a mixture of uppercase and lowercase.
- The `callback` parameter contains a random string that we will need later.
- The `show` parameter allows Rapportive to specify what information should be returned. The parameter is omitted on a normal lookup, but some special values are defined – see below.

Some values may be unknown for some people. If a parameter is missing or its value is the empty string, your Raplet should simply treat it as unknown. The parameters may appear in any order, and parameters may be added or removed in future. If your Raplet doesn't recognize a parameter, it should ignore it.

For example, if your Raplet is at `http://localhost:8080/raplet.php`, your web browser will request something like the following:

```
http://localhost:8080/raplet.php?email=rahul@rapportive.com&name=Rahul%20Vohra&twit
ter_username=rahulvohra&callback=jsonp123456789
```

RESPONSE FORMAT

Ok, your server has received a Raplet request; you have read out the parameters, queried your database and maybe found something relevant about that person. Now you build your response to be sent back to the browser.

The response must be formatted as a JSON object. The following properties are currently supported in the response object:

- `html`: The contents of your Raplet for the person you just looked up, formatted as a HTML string. Do not include any `<style>` or `<script>` tags – use the `css` and `js` properties instead. If your Raplet doesn't have anything useful to display, please return an empty string.
- `css`: String containing CSS stylesheet rules for formatting the Raplet HTML. The rules are automatically scoped to your Raplet, so you don't need to worry about interfering with the styling of the rest of Gmail or other Raplets. Please include all CSS here and do not use @import. Your CSS should be W3C valid and not use browser hacks (invalid CSS will cause undefined behavior). Your Raplet should always return the same CSS string, no matter which person is being looked up, since Rapportive may cache the CSS.
- `js`: String containing the Raplet's JavaScript code. Please see the section on advanced usage below.
- `status`: An integer with a meaning similar to the HTTP status code. We use this code because we can't use the real HTTP status code in conjunction with JSONP. For example, if your Raplet successfully finds information, the value of the `status` property should be `200`. If your Raplet can't find any information about the person, `status` should be `404`. Your real HTTP status code should be `200` in any case.

All of this must be encoded as a JSON object (see the specification at http://json.org for details). Your JSON must be valid (invalid JSON causes errors which are hard to diagnose), so we strongly recommend that you use a standard JSON library that escapes everything correctly. JSON libraries are available for all mainstream programming languages.

Finally, when you have encoded your JSON, there's one last thing you need to do. Before the JSON object, you need to prepend the value of the callback query parameter (e.g. `jsonp123456789`) and an opening parenthesis '`(`'. After the JSON object, you need to append a closing parenthesis ')'. And that's your entire response. It should look something like this:

```
jsonp123456789({
    "html": "<h2 class='name'>Rahul Vohra</h2><p>Lifetime value: $1,000,000!</p>",
    "css": "h2.name { color: grey }",
    "js": "alert('Hello world!');",
    "status": 200
})
```

Make sure there is no spurious HTML anywhere in the response, otherwise your Raplet will not work. Some programming environments (e.g. PHP) print warning messages directly to the page, which will cause your response to be invalid, so please ensure that all errors and warnings are handled properly. If you can, you should send your response with Content-type `text/javascript`, although most browsers will accept your response even if it's a different content type.

ADD THE RAPLET TO RAPPORTIVE

Open Gmail in a browser with Rapportive installed. Click “Rapportive” at the top of the Gmail page, then click “Add or Remove Raplets”. You will be logged in to Rapportive if necessary. (If you are already logged in, you can take a short cut by going directly to http://raplets.com.)

On the Raplets page, click “Add” in the “Custom Raplet” section, and paste your Raplet URL into the field provided. In this example it would be `http://localhost:8080/raplet.php`.

Go back to Gmail and look up a person in Rapportive, e.g. by hovering over an email address. The sidebar should load, including your Raplet. It’s usually not necessary to reload your Gmail tab, but if it’s not working as expected, try reloading Gmail as your first measure.

If it doesn’t work, please see the Troubleshooting section below. If it does, congratulations on your first Raplet! If you want it to be added to our directory of featured Raplets on the Rapportive website, please contact support@rapportive.com.

ADVANCED USAGE

The best Raplets provide rich interactive behavior that allows users to accomplish tasks without ever having to leave their inbox. You can make your Raplets interactive using JavaScript.

JAVASCRIPT AND JQUERY

You can include JavaScript in your Raplet as a string in the `js` property of the JSON response object. As with all other content, please make sure that it is correctly escaped for JSON. The Raplet’s JavaScript code is executed after the Raplet HTML has been inserted into the page, and every time the sidebar is refreshed.

jQuery is automatically provided for you (currently version 1.4.1), and we strongly recommend that you use it. You can use jQuery’s functions `jQuery` and `$`. They are automatically scoped to your Raplet: for example, `$('a')` selects all links within your Raplet, not all links in the entire Gmail page. Please see the section on security below for further detail.

For example, a Raplet response with JavaScript may look a bit like this:

```
jsonp123456789({
    "html": "<h2 class='name'>Rahul Vohra</h2><p>Lifetime value: $1,000,000!</p>",
    "css": "h2.name { color: grey }",
    "js": "$('h2.name').click(function(){ $(this).text('You clicked me!'); });",
    "status": 200
})
```

SENDING DATA TO YOUR SERVER

To provide interactive features, your Raplet may need to send information back to your server (e.g. if you want to provide a button which adds a new contact to your CRM). You would normally do this using Ajax, but the standard techniques and libraries that rely on `XMLHttpRequest` do not work in Raplets. This is because the browser page was loaded from `mail.google.com`, and for security reasons, browsers block cross-domain requests to any server other than `mail.google.com`.

We currently have two methods which you can use to send data back to your server: JSONP and IFrames.

- JSONP is the technique Rapportive uses to load the sidebar and Raplets, so you already know what it looks like. jQuery makes it easy to send JSONP requests from your Raplet's code: please see http://www.ibm.com/developerworks/library/wa-aj-jsonp1/ for an introduction. The advantage of JSONP is that gives you the greatest flexibility; its disadvantages are that you can only make `GET` requests (no `POST`), you cannot send more than about 2kB of data to your server, and it can introduce security problems unless you are careful. Please see the section on security below.
- IFrames allow you to embed a completely different page within Gmail. You can include an `<iframe>` tag in your Raplet HTML, and load a page from your server into it. You will probably want to include query parameters in the IFrame source URL, so that it knows the context in which it is being displayed. Within that frame you can use standard HTML forms and Ajax to send data back to your application. However, there are disadvantages: the IFrame has a fixed size which cannot easily be changed from inside the IFrame; code inside the IFrame cannot access any of the JavaScript APIs provided by Rapportive.

CACHING

At the time of writing, Raplets are not cached by Rapportive: every time a Raplet is displayed, it is freshly fetched from your server and any JavaScript is executed. However, to improve performance we will cache Raplets in future. You should design your Raplet to be cacheable now, so that you don't need to change it when caching is introduced.

When Rapportive loads your Raplet for two different people, we assume that the `status` and `html` properties may differ, but we expect the values of the `css` and `js` attributes to be identical for different people. Your CSS and JavaScript still apply when your Raplet is loaded for a new person, but we may reuse the previous lookup's CSS and JavaScript to avoid having to re-evaluate it every time.

If you need to access any person-specific data in your JavaScript code, please encode it in the HTML, and use jQuery's DOM inspection methods to figure out how the code needs to behave. For example, say you want to trigger a different handler depending on whether a contact is already in your database. You can use a CSS class to denote whether a person is known to your database:

```
<button class="person known">Edit Bob</button>    <!-- Bob is in our database -->
<button class="person unknown”>Add Fred</button>  <!-- but Fred isn’t -->
```

Then your JavaScript can handle both cases:

```
    $('button.person').click(function() {
        if ($(this).hasClass('known')) {
            // Edit this person in our database
        } else if ($(this).hasClass('unknown')) {
            // Add this person to our database
        }
        return false;
    });
```

Now it is sufficient for Rapportive to fetch your CSS and JavaScript once in a while, and to reuse the code between Raplet lookups. So we can make further improvements by sending the CSS and JavaScript over the network only if Rapportive needs to update them. Rapportive will manage the caching; if no new CSS and JavaScript are needed, Rapportive will include the query parameter `show=htmlonly` in the Raplet request URL. If your Raplet sees this parameter, it should omit the `js` and `css` properties in the response object, and only include the `html` and `status` properties. If you send `js` and `css` properties anyway, Rapportive may choose to ignore them, and use the cached values instead.

METADATA AND CONFIGURATION

There is more information surrounding Raplets than just looking up people. Unless you are the only person using your Raplet, you probably want to give it a name, description, icon and other information that helps users make the most of your Raplet.

In addition, some Raplets need to be configured by the user. For example, if you are building on a third-party API which requires that users provide their API key or password, you need some way of asking the user for that information when they install your Raplet. This is achieved with the metadata protocol.

If you submit your Raplet to Rapportive to be included in our list of featured Raplets, the information about your Raplet will also be taken from this metadata.

THE METADATA PROTOCOL

When your Raplet is configured, Rapportive may ask your server for additional information about your Raplet. This is done with a request to your Raplet URL, but instead of the `email`, `name` and other person-specific parameters, the following query parameters are given:

- The `show` parameter is set to the fixed value `metadata`. This is how you identify a metadata request.
- The `callback` parameter contains a random string, as in the other types of lookup.

For example, a metadata request may look like this:
`http://localhost:8080/raplet.php?show=metadata&callback=jsonp123456789`

If your Raplet receives a metadata request, instead of the usual Raplet response, it should return a JSONP object with the following properties (all required unless marked “optional”):

- `name`: The name of your Raplet, which will be used as a label in various places. Should be short (less than 15 characters, if possible) and cannot contain HTML.
- `description`: A few words (up to about 100 characters) describing what your Raplet is about. HTML is permitted, but no block elements please (it’s ok to make a word italic, for example).
- `welcome_text`: A longer text (up to a few paragraphs) introducing your Raplet and giving instructions to users who want to install it. HTML is permitted. This text is displayed in a dialog box when a user clicks to install your Raplet.
- `icon_url`: The absolute URL (starting with `http`) of an icon to represent your Raplet. It should be in PNG or GIF format, 100x100 pixels large, with a transparent background. It should be designed to look good on a white or light grey background, and it should not include a border (we take care of the border and rounded corners). Example: http://rapportive.com/images/raplets/crunchbase-icon.png
- `preview_url`: The absolute URL (starting with `http`) of an image containing an example screenshot of your Raplet. It is used on http://rapportive.com/raplets to give users an idea of what their sidebar looks like. You may choose the height of the image, but it must be 220 pixels wide, on white background, with equal margins on either side. Example: http://rapportive.com/images/raplets/crunchbase-preview.png
- `provider_name`: Your name or the name of your company, so that users can see who is providing a Raplet. Cannot contain HTML.
- `provider_url`: The URL of your website or your company’s website, so that users can learn more about you, look up your privacy policy and check that they are comfortable with installing your Raplet.
- `data_provider_name`: (optional) If you are using a third-party data provider or API in your Raplet, please include their name here. Cannot contain HTML.

- `data_provider_url`: (optional) If you have a `data_provider_name`, include the URL of their website here.
- `configuration`: (optional) An array of configuration parameters, documented in a separate section below.

For example, your response may look something like the following:

```
jsonp123456789({
    "name": "CrunchBase",
    "description": "Essential intelligence for every entrepreneur and investor. ",
    "welcome_text": "<p><a href=\"http://www.crunchbase.com\">CrunchBase</a> is an
essential resource for tech entrepreneurs and investors.</p>\n<p>...</p>",
    "icon_url": "http://rapportive.com/images/raplets/crunchbase-icon.png",
    "preview_url": "http://rapportive.com/images/raplets/crunchbase-preview.png",
    "provider_name": "Rapportive",
    "provider_url": "http://rapportive.com/",
    "data_provider_name": "the CrunchBase API",
    "data_provider_url": "http://www.crunchbase.com/help/api"
})
```

CONFIGURATION PARAMETERS

If your Raplet needs to be configured, you can specify a `configuration` property in the metadata object. When a user installs your Raplet, the `configuration` property is translated into a form on the install dialog, and Rapportive captures the user's input on your behalf. Rapportive stores the user's settings and includes the values as query parameters on every Raplet request.

The `configuration` property should contain a JSON array, each element of which is a JSON object with the following properties:

- `name`: The name of the query parameter you want to use for this setting. We recommend that you limit yourself to alphanumeric and underscore (_) characters, and avoid any parameter names already used by Rapportive.
- `description`: The label text you want to display for this form field, explaining what the setting is about. HTML is permitted (we generate the `<label>` tag, you provide the contents).
- `type`: Tells us what kind of input control you want for this settings. Currently accepted values are `text` (a single-line text field) and `password` (a password field which masks its input).
- `required`: Boolean; if set to `true`, the user is not allowed to leave this setting blank. This property is optional and defaults to `false`.

For example, a metadata response with configuration parameters may look as follows:

```
jsonp123456789({
    "name": "Transmogrifier",
    /* ... other metadata properties omitted for readability ... */
    "configuration": [
        {
            "name": "magic_token",
            "type": "text",
            "description": "Your magic token (please copy and paste from <a
href=\"http://example.com/account\">your account page</a>)"
        }, {
            "name": "secret",
            "type": "password",
            "description": "Your secret incantation",
            "required": true
        }
    ]
})
```

We ask you to keep the configuration as minimal and simple as possible. (The fewer settings a user can change, the less confusing they will find the experience of adding your Raplet, and the fewer mistakes they can make.) Only ask for things for which you cannot provide defaults, such as login details or user-specific API keys.

SECURITY

OPENNESS OF JSONP

Rapportive uses JSONP to load Raplets because it enables cross-domain requests (so that a page from `mail.google.com` can make a request to your Raplet server, and receive the response). This feature is also JSONP's greatest weakness: not only Rapportive and pages on `mail.google.com`, but pages on **any domain** can make requests to your Raplet URL.

This creates a security risk if your Raplet provides access to sensitive data: if an attacker knows your Raplet URL and they can trick you into visiting a specially prepared web page (e.g. by sending you an email like "hey, look at my funny pictures of cats on this site!"), they can use your browser to send requests to your Raplet, leak the data back to their own servers and thus potentially gain access to your sensitive information.

You can prevent this kind of attack using a secret token, as follows:

1. For each user of your application who may use a Raplet, generate a "token", which is simply a string that is sufficiently random that it cannot be found by guessing or brute force. (If you already have a per-user secret API key, authentication token or similar, you may reuse that.)
2. Use a configuration parameter (see above) to ask users to copy and paste this token from your application into Rapportive when they add your Raplet. Make the setting required.
3. We store this token for each Rapportive user on Rapportive servers, and include it as a query parameter on all requests to your Raplet.
4. When your Raplet receives a request for data, you check that a token has been included as a query parameter, and that it matches the token you generated for that user. If the token is missing or does not match, you reject the request and provide a helpful error message.

Your Raplet is now secure because the attacker cannot detect your users' tokens. Metadata requests are not protected by tokens, but they shouldn't contain any substantial sensitive information anyway.

The user experience of copying and pasting tokens sucks, though — it is error-prone and confusing to users. We are planning to support OAuth in future to make this better.

JAVASCRIPT SANDBOXING

In the current version of Rapportive, JavaScript inside Raplets is run with only minimal sandboxing: exceptions are caught, and jQuery selectors are by default scoped to the DOM element containing the Raplet HTML.

A malicious Raplet author could easily circumvent these limitations and access any part of the Gmail page, which may contain large amounts of sensitive information. We are therefore warning users that they should only install Raplets from sources they trust.

We recognize that trust alone is not enough, so we are planning to fully sandbox Raplets in future. By sandboxing, we will eliminate all known ways of accessing or modifying data to without the user's explicit permission. We are also planning to introduce additional APIs through which Raplets can request permission to access information other than the parameters we pass to the Raplet request.

To avoid breaking your Raplet when we introduce sandboxing in future, we ask you to write your JavaScript very conservatively:

- Do not rely on global browser objects such as `document` or `window`. Always use jQuery to find and manipulate DOM elements.
- Do not use `eval` – there are very few cases in which it is legitimate.
- Be very careful with `setTimeout` and `setInterval`; if you really must use them, pass them a function object, not a string.
- Do not place `<script>` tags in the HTML for your Raplet; use the `js` property instead.
- Do not use `onclick` attributes or `javascript:` URLs in HTML; bind functions to events using jQuery instead.

This is not an exhaustive list of limitations. In general, if you write your JavaScript according to best practices, it should continue working when we introduce sandboxing of Raplets.

PRIVACY

By observing the email addresses and other information sent to your Raplet, you could learn a large amount of potentially sensitive information about a user. We expect you to be absolutely transparent and honest about how you will use and safeguard this information, and it is up to you to make sure that users of your Raplet understand your policies.

If you request that your Raplet is added to the list of suggested Raplets on the Rapportive website, we will need to periodically verify that your handling of data meets Rapportive's legal and ethical standards.

CODE SAMPLES

Coming soon…

TROUBLESHOOTING

DEBUGGING YOUR RAPLET

If your Raplet does not work, diagnosing what is wrong is not straightforward. Here are some suggestions:

- Check your web server logs to see if Rapportive is making requests to your Raplet URL. If not, check that you have entered the correct URL at http://rapportive.com/raplets, and that you are logged in to Rapportive with the same browser as you used to add your Raplet. You may need to reload your Gmail tab for Raplet changes to take effect.
- Check your Raplet server's response by requesting the Raplet URL (with query parameters) in a separate browser tab. Make sure that it is valid JSONP (no spurious text inserted, no escaping errors), and that it has the right properties as described in this document.

- Open the Firebug Console tab (on Firefox) or the JavaScript Developer Console (on Chrome). Do any error messages appear?
- Open the Firebug Net tab (on Firefox) or the Resources tab in Developer Tools (on Chrome), and look for the JSONP calls to your Raplet. Can you notice anything unusual? Note that enabling resource tracking will slow down your Gmail, but it can be very useful for tracking down some tricky cases.
- If nothing else helps, try the sledgehammer of web development: sprinkle `alert()` calls around your JavaScript… it's not pretty, but still remarkably effective.

THIRD-PARTY COOKIES

Some of your users may have third-party cookies disabled. If your Raplet sets cookies — for example, after authenticating a user — then these users will either have to enable third-party cookies, or they will have to add an exception for your server's hostname.

SUPPORT

We would like to invite you to share your experiences and thoughts on our raplet-dev mailing list. This is also a good place to ask any questions you have or get help if you get stuck. Please sign up for the list at http://groups.google.com/group/raplet-dev.

Please also make sure that you subscribe to the raplet-announce list. This is a low-traffic list on which we post updates that affect all Raplet developers. Please join it at http://groups.google.com/group/raplet-announce.

APPENDIX

CHANGELOG

Version 0.3 (May 2010)

- Added support for the Raplet metadata protocol and configuration settings
- Documented security measures and best practices for Raplet development
- Added `status` property to Raplet response object
- Specified that `js` and `css` properties must not vary by person, as they may be cached
- Added lots of detail throughout the document

Version 0.2 (April 2010) — first public release

- Added `name` and `twitter_username` to Raplet query parameters
- Added `js` and `css` properties to Raplet response object; deprecated inline `<script>` tags

Version 0.1 (March 2010)

- Raplets first introduced to a small number of alpha testers