# Transparency & Consent Framework

## *Consent Manager Provider JS API*
## Draft for Public Comment
## v1.0

## Table of Contents

# Introduction

In February 2017, the IAB Europe assembled parties representing various participants in the online advertising ecosystem, in particular parties from both the supply and demand side of the ecosystem, to work collectively on guidance and solutions to the requirements of the General Data Protection Regulation (the "GDPR"). That working group is currently known as the GDPR Implementation Working Group (or the "GIG")). One of the working groups within the GIG was tasked with developing guidance on consent as a legal basis of processing. An additional working group developed out of that working group to develop a standard solution for companies to use, where and if necessary, to request, obtain and disseminate consent to various parties in the online advertising ecosystem that may be relying on consent as a legal basis of processing and/or may have parties integrated with them that rely on consent.

This document is a draft for public comment. Please submit your general feedback to feedback@advertisingconsent.eu and any technical feedback to transparencyframework@iabtechlab.com by April 8, 2018.

## About the Transparency & Consent Framework

The scope of the working group's initiative increased further into a broader initiative to develop an industry solution to

1. (allow website operators to control the vendors it wishes to allow to access to its users' browsers/devices and process their personal data and disclose these choices to other parties in the online advertising ecosystem;
2. seek user consent under the ePrivacy Directive (for setting cookies or similar tech and accessing info on a device) and/or the GDPR in line with applicable legal requirements and disseminate the consent status through the online advertising ecosystem;
3. have one place to go to
    - (a) understand privacy-related disclosures about those vendors;
    - (b) use those disclosures to make privacy-related disclosures to its users generally;
    - (c) provide disclosures required to be provided by vendors that are Controllers to end users;
    - (d) seek user consent in line with applicable legal requirements where vendors may require it under the ePrivacy Directive and/or GDPR for various purposes, and
    - (e) disseminate the consent status through the online advertising ecosystem.

The various pieces of the framework are the following:

- A global vendor list
- The reference architecture (for cookie format and vendor list and related API's)
- Policy underlying
    - the disclosures to be made by vendors included on the global vendor list
    - the use of the global vendor list and the reference architecture

The transparency and consent framework and the various standards introduced by it, including the standard detailed below, are a work-in-progress and currently designed to be used for testing.

## About the Transparency & Consent Standard

For purposes of this documentation, the following terms have the following definitions:

- "***CMP***" means a company that can read the vendors chosen by a website operator and the consent status of an end user (either service specific (through a first-party cookie) or global (through a third-party cookie). A CMP is not synonymous with a company that surfaces the user interface to a user (although it can be the same).
- "***Purposes***" mean the purposes for which a Controller enabled by a website operator is using personal data collected from (or received by a third party) about an end user.
- "***Daisybit***" means information compressed into a binary value and passed throughout the online advertising ecosystem through the OpenRTB specification.
- "***Vendor***" means a third party that a website operator is using in connection with surfacing content to its end users that either (1) accesses an end user's device or browser; and/or (2) collects or receives personal data about the website operator's end users. As such, a vendor need not be a Controller.

## License

# CMP JS API v1.0

## Version History:

v1.0 - Initial version published December 19, 2017

## What is supported by this API?

This API draft takes the approach of specifying the minimum-necessary functionality that the CMP needs to provide DSP's and SSP's vendor consent info. There's a large potential surface area of publisher-CMP functionality (including UI control and configuration) that is best provided by CMP-specific, rather than standardized, API's.

## What API will need to be provided by the CMP?

Every consent manager will provide the following API

- **__cmp(*Command, Parameter, Callback*)**

In addition, an event handler will handle postMessage events with a *__cmpCall* key in the event data (see below), for calling the CMP from iframes.

This API has to support the following functionality.

| **Command: String** | **Parameter** | **Callback function signature** | **Comments** |
|---|---|---|---|
| Required call to be implemented by CMP | | | |
| getVendorConsents | vendorIds: Uint16Array | Callback( *VendorConsents object)* | The vendorIds lists the vendor ids for which consent is being requested.<br>The callback function will be called with a *Consents* object as the parameter, the keys being the vendor ids requested.<br>If vendorIds is null or empty, the *Consents* object will be empty ({} object).<br>The consent will be returned false ("No Consent") for any invalid vendorId. |

| | | | |
|---|---|---|---|
| getConsentData | type: string ("vendorConsents") | Callback(*consent data string)* | Returns the requested cookie value as a websafe base64-encoded string. |
| Optional calls to be implemented by CMP | | | |
| getPublisherConsents | purposeIds: Uint16Array | Callback( *PublisherConsents object*) | The purposeIds lists the purpose ids the publisher is requesting consent for. If this array is null or empty, it will default to all configured purposes.<br>The callback function will be called with a *Consents* object as the parameter, the keys being the publisher purpose ids requested.<br>The purpose ids would be set by the publisher using a CMP-defined initialization function. |
| getVendorList | VendorListVersionId | Callback(vendor list object) | The callback function will be called with a JSON object representing the vendor list for the requested version id as the parameter.<br>If the VendorListVersionId is null, then the version list for version in the cookie is returned. If no cookie was set, then the latest version of the list is returned.<br>If the VendorListVersionId is invalid, callback function will be called with null object as argument. |

## What are the VendorConsents and PublisherConsents objects?

The *VendorConsents* object contains the global purposes consented, and the vendors consented to as key:value pairs:

```
{

  metadata: websafe-base64 encoded string,
  purposes: {
    purposeId: consentBoolean,
    …
  },
  vendorConsents: {
    vendorId : hasConsentBoolean, …
  },
}
```

where *vendorId* and *purposeId* are the keys and *consentBoolean*, *hasConsentBoolean* are the values for the consent (false="No Consent", true="Consent"). The `metadata` will have the following information as described in the cookie format.

1. Cookie Version
2. Created Timestamp
3. Last Updated Timestamp
4. Cmp Id
5. Vendor List Version

The *PublisherConsents* object contains the publisher-specific purposes consented to:

```
{
  standardPurposes: {
    purposeId: consentBoolean,
    …
  },
  customPurposes: {
    customPurposeId: consentBoolean,
    …
  }
}
```

## How will the components calling this API check if the CMP script has loaded yet?

Typically, they will not need to. The components will be able to call the API without having to check if the CMP script has loaded, as calls will be queued by a stub implementation and called when the CMP is loaded. If this is necessary, a property could be set on the stub and/or the actual implementation of **__cmp** to distinguish them.

## How can __cmp to be called before the CMP script has loaded?

The prerequisites are:
1. The CMP tag (that the publisher will add to their page) will:
1.1. Defines a stub function **__cmp()**
1.2. This stub stores all the calls to this function and the parameters made before the CMP loads.
1.3. Tag is added to the header of the page
2. The CMP API script will do the following when loaded:
2.1. The *__cmp()* function will be set to the CMP's full API implementation.
2.2. It then executes any calls queued by the stub (by iterating through the array of commands stored by the stub), in the order received.

### Is there a sample CMP stub?

```
<!-- Consent Manager; In the example, the cmp.js is loaded directly. this js could be loaded
asynchronoulsy via the tag as well-->
<script type="text/javascript" src="<%= somepath %>cmp.js"></script>
 <!-- Consent Manager Tag : Stubbed -->
 <script type="text/javascript" , async=true>
  var __cmp = (function () {
    return typeof (__cmp) == "function" ? __cmp : function (c) {
      var b = arguments;
      if (!b.length) {
        return __cmp.a;
      }
      else if (c == '__cmp')
        return false;
      else {
        if (typeof __cmp.a === 'undefined') {
          __cmp.a = [];
        }
        __cmp.a.push([].slice.apply(b));
      }
    }
  })();
</script>
<!-- End Consent Manager Tag : Stubbed -->
```

## How can vendors that use iframes call the CMP API from the iframe?

The postMessage() function can be used to proxy calls to __cmp() . CMP's will establish an event handler to call *__cmp()* for any postMessage events with a *__cmpCall* key in the data, returning the data via a postMessage event with data with a *__cmpReturn* key.

For example:

```
// CMP needs to know the id of the iframe, so that it could post
// the result back to the iframe.
var iframeRegisterToCMPMsg = {
  __cmp: {
    command: 'registerIframeVendor',
    parameter: {"id": iframeId},
    callId: uniqueId,
  }
}
parent.postMessage(iframeRegisterToCMPMsg, '*');

// call CMP to get consent data
var messageToCMPWindow = {
  __cmpCall: {
    command: 'getVendorConsents',
    parameter: {"vendorIds" : [vendorIds]},
    callId: uniqueId,
  }
}
parent.postMessage(messageToCMPWindow, '*');

// Event handler for response
window.addEventListener("message", function(event) {
  // if event.data.__cmpReturn exists,
  // event.data.__cmpReturn.returnValue will be the return data
  // event.data.__cmpReturn.callId will be the (unique) call id
}, false);
```

Values of *command* and *parameter* in the sent message are the same as those arguments when calling __cmp().

On the other side, the CMP needs to have an event listener listening for messages from ad vendors, and register a message processing function with the event listener, for example:

```
// call registerIframeListener() in initialization
var registerIframeListener = function() {
  if (window.addEventListener) {
    window.addEventListener('message', iframeAdMsgHandler, false);
  } else {
    window.attachEvent('onmessage', iframeAdMsgHandler);
   // for IE
  }
};

// the method of message processing
var iframeAdMsgHandler = function(event) {
  // 1. deal with message processing
  // 2. Call the CMP API
  // 3. Post the message back to the ad iframe

  // Pseudocode:
  var json = JSON.parse(event.data);
  switch(cmd) {
    case 'getPublisherConsents':
      Result = getPublisherConsents();
      Break;
    // etc, etcc
   }

   // find the iframe by its id
   var iframeWindow = document.getElementById(iFrameId)
                              .contentWindow
   iframeWindow.postMessage(JSON.stringify(result), iFrameOrigin);
}
```

In case of multiple ad iframe on the publisher site, CMP should be able to store the domain and the id of the iframe, so it could post the message back to the right iframe.

## Where will the API retrieve the vendor consent information from?

CMP implementations will retrieve vendor consent from the third party global cookie, or if configured and implemented by the CMP, a first-party service-specific cookie (or, for service-specific vendor consent that needs to be shared between sites, a shared third-party cookie location).

## How will the API prioritize the service-specific and the global cookies?

The prioritization between these two cookies is as specified in the policy FAQ.
The service-specific cookie consents will override the global consent cookie, if it is being used. These are the following resolutions of consent for each vendor:

| Service-specific Consent | Global Consent | Service-specific cookie being used? | Resultant Consent |
|---|---|---|---|
| No | don't care | Yes | No |
| Yes | don't care | Yes | Yes |
| don't care | No | No | No |
| don't care | Yes | No | Yes |

## Are cookies a must for this API?

For the current version: Yes
For future versions: Not necessarily.
The current version does depend on the the consent data being stored in cookies. In the future, the solution could be moved to use other consent storage mechanisms like central registries storing user id and their consent information. At that time the implementation will need to be updated to retrieve consent data from the other source. However the API interface itself need not change.