# Preference Center Application Programming Interface (API)

# Functional Specification

# Section 1

## Introduction

Channels of communication with contacts shall be supported for SMS, email, catalogue and reminders. Future expansion for additional channels will be supported. Frequency of communication shall be configurable at the brand and channel level. Frequency is customizable to support daily, weekly, monthly etc. Ranking of frequency is supported at the communication channel level. Ranking of brands is supported at the enterprise level.

Being the first REST web services project presents an opportunity to build in at a basic level, tried and proven industry standard fundamental best practices as well as forward looking features and technologies. These could perhaps be replicated in other REST web services, or if the current Web Service which is being developed is extended in the future, then its extensions would also have the available features.

This API will essentially be an interface to the database (IBM DB2) and will not perform any calculation or data manipulation. However, validation of input and out data will be performed.

## Purpose and Contents

This document provides outline functional specifications and requirements for Web Services (Non-database specific) portion of the 1800flowers Preference Center Application Programming Interface (API).

It is designed to guide system development and design, including:

* Web service architecture
* Various functionalities
* Key class structures and field constraints

## Requirements

## Administration Resources

1. The following basic resources have been defined and access to this data will be provided by the DBA via stored procedures.

* Contact
* Brand
* Communication Channel
* Frequency

1. The following derived resources have also been defined and access to these will also be provided by the DBA via stored procedures.

* Brand Communication
* Communication Channel Frequency

1. Web Services are by definition stateless so there will be no security features to restrict access to the service or the data provided. It will then be the responsibility of MDM to ensure that the user is registered and authenticated.
2. DBA will provide all data manipulation and retrieval logic via stored procedure or otherwise.

# Application Programming Interface (API)

This section describes the request and response interfaces for the preference center API.

* JSON data over a REST implementation will be used.
* Resources shall be depicted by following a Uniform Resource Identifier (URI) construct.
* The preference center API shall be identified by the following URI prefix for all method calls
  + https://casregservices.1800Flowers.int/api/preferencecenter

## Uniform Interface (URI)

HTTP provides four basic methods for the four most common operations. All four will be used for the preference center implementation:

* Retrieve a representation of a resource: HTTP GET
* Create a new resource: HTTP PUT to a new URI, or HTTP POST to an existing URI (i.e. adding to a collection).
* Modify an existing resource: HTTP POST to an existing URI. This shall be idempotent.
* Delete an existing resource: HTTP DELETE. This shall be idempotent.

### API Operations

Please see Appendix C for the detailed description of the operations that will be included within the preference center API.

NOTE: [host] refers to <https://casregservices.1800Flowers.int/api/preferencecenter>

# Design Patterns

This section outlines the design patterns that will be used in the implementation of the web service:

* DAO – Data Access Object – for interacting with the database.
* Business Delegate - to decouple the API from the specific database implementation while simultaneously allowing for a highly cohesive design.
* DTO - Data Transfer Object – to be used in processing data obtained from the DAO.
* Value Object – These objects will represent the data as it will be presented to the consumer.
* Façade – Groups closely related functionality together for a simpler unified interface
* Helper – Helper classes will be used.

# Considerations

This section describes considerations that will be integral to the design of the web service:

* The web service will be built from scratch and will not use any pre-existing web service framework or reference implementation currently at 18F.
* Large emphasis will be placed on error handling.
* The input requests will be carefully validated and the appropriate response code will be returned. (See Appendix A.)
* The Web Service API will be implemented to accommodate Swagger integration.
* IBM SoftLayer sandboxing will require design that has a small software footprint.
* Extensibility and maintainability will also be key considerations.
* Implementation will be based on Java JDK 1.6 as the application needs to be able to be deployed on existing 18F server architecture.
* Useful, fine grained and configurable logging functionality will be integrated.

# Best Practices

This section describes general industry best practices when designing and implementing web services:

* It should use web standards where they make sense.
* It should be friendly to the developer and be explore-able via a browser address bar.
* It should be simple, intuitive and consistent to make adoption not only easy but pleasant.
* It should provide enough flexibility to be useful in different scenarios.
* It should be efficient, while maintaining balance with the other requirements.

# Additional features

This section describes features that will also be developed:

* Logging mechanism/framework log4J2 will be used to manage application logging.
* Email notification of critical errors will also be available.
* Apache Commons Validation framework will be used to validate input/output data at the field level.
* API will be designed to not suppress errors, but will allow exceptions to ‘bubble up’ to the higher classes where they can be used to customize the web services responses.
* Deployment configuration will be embedded in the ear file to assist in simplifying the deployment process.
* If the database encounters anomalies then the web service will send the error message to the response.
* There will be a general container level exception handler so the application can respond to any request and if needed return a graceful error message response.
* All responses will be JSON formatted.
* Logging (log4J2.xml) configuration file deliberately placed outside the ear file so as to provide an opportunity to change production logging settings in the event that changes to log settings is needed.
* Error messages will be as specific as possible.

# Key Libraries Used

This section describes important libraries to be used:

* IBM Wink REST container will be used.
* J2EE servlet-api (version 2.5)
* Jackson Annotations module.
* Apache Commons Validation.
* Faster XML implementation of Jackson will be used.
* DB2 JDBC driver will be db2jcc.
* Apache BVal Bean Validation Provider.
* JSR311 implementation of REST Web Service will be used.
* log4J2 for application logging.
* Java-Mail

# JSON Responses

This section describes the JSON response structures that are created:

## Error:

[

{

"message\_order": 0,

"message\_type": "ERROR",

"message\_format": "TEXT",

"message": "Exception encountered processing data records with message: No records were returned by database for the search parameters specified.."

}

]

## Status:

[

{

"statusCode": 403,

"statusDescription": "Forbidden",

"statusMessage": "When authentication succeeded but authenticated user doesn't have access to the resource"

}

]

# Available utilities

This section describes useful available utilities:

|  |  |  |
| --- | --- | --- |
| **Tool** | **Description** | **Comments** |
| ping | This is a ping service call feature which may be used to verify if the web service is up. It also attempts to connect to the database. | /tools/ping |
| wadl | This is a service call that returns the current xml formatted WADL file representation of the web service. | /tools/wadl |

# Security Considerations

This section describes important security features that will be integrated:

* Even though the web service does not perform authentication/authorization, security will be a core design time concern.
* Http methods that are not going to be used will be actively blocked by the servlet container.
* All access to the database will be done via JDBC prepared statement database calls.
* There will be no dynamically constructed queries.

### Appendix A

# Response Codes

This section describes the response codes that will be used. These are industry standard response codes normally used by web service applications:

### Standard Response Codes

|  |  |  |
| --- | --- | --- |
| Code | Response | Comments |
| 200 | OK | Response to a successful GET, PUT, PATCH or DELETE. Can also be used for a POST that doesn't result in a creation. |
| 201 | Created | Response to a POST that results in a creation. Should be combined with a Location header pointing to the location of the new resource. |
| 204 | No Content | Response to a successful request that won't be returning a body (like a DELETE request). |
| 304 | Not Modified | Used when HTTP caching headers are in play. |
| 400 | Bad Request | The request is malformed, such as if the body does not parse |
| 401 | Unauthorized | When no or invalid authentication details are provided. Also useful to trigger an auth popup if the API is used from a browser. |
| 403 | Forbidden | When authentication succeeded but authenticated user doesn't have access to the resource. |
| 404 | Not Found | When a non-existent resource is requested. |
| 405 | Method Not Allowed | When an HTTP method is being requested that isn't allowed for the authenticated user. |
| 410 | Gone | Indicates that the resource at this end point is no longer available. Useful as a blanket response for old API versions. |
| 415 | Unsupported Media Type | If incorrect content type was provided as part of the request. |
| 422 | Unprocessable Entity | Used for validation errors. |
| 429 | Too Many Requests | When a request is rejected due to rate limiting. |
| 500 | Internal Server Error | The server has encountered a situation it doesn't know how to handle. Please contact the web service(s) system administrator. |
| 501 | Not Implemented | The request method is not supported by the server and cannot be handled. |
|  |  |  |

### Custom Response Codes

|  |  |  |
| --- | --- | --- |
| Code | Response | Comments |
| 554 | Bad input for create | Bad input data for create request. |
| 555 | Bad input for update | Bad input data for update request. |
| 556 | Bad output data | Bad output data in response object. |
| 557 | Bad input data | Bad data in request. |
| 561 | Delete failed | Unable to delete data as requested. |
| 563 | Resource unavailable | Unable to locate requested resource. |
| 562 | Update Failed | Unable to update resource as requested. |
| 564 | Create Failed | Unable to create resource as requested. |
| 567 | Invalid Data Returned | Invalid Data Returned from database. |
| 568 | Invalid URI constructed | Invalid URI constructed |
| 569 | Missing resource bundle. | Missing resource bundle. |
| 570 | Unable to complete sort request. | Unable to complete sort request. |
| 571 | Invalid OS | Unable to run service on target operating system. |
| 572 | Unable to connect to database | Unable to establish database connection. |
| 573 | General database error | General database error. |
| 576 | No records found | No records were returned by database for the search parameters specified. |
| 574 | General error | General application error non-specified. |
| 575 | Invalid request | Invalid request. |

### Appendix B

# Class Design

This section describes the java classes that will represent the resources to be used in this project:

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Constraint** | **Comments** |
| description | String | Size: 0 and 50 characters |  |
| rank | integer | Min Value=0 |  |
| code | String | Not null  Size: 1 and 5 characters |  |
| active | Char | [‘Y’,’y’,’N’,’n’,’1’,’0’] | Allow only one character |
| visible | Char | [‘Y’,’y’,’N’,’n’,’1’,’0’] | Allow only one character |

**Communication Channel (Channel.java)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Constraint** | **Comments** |
| description | String | Size: 0 and 50 characters |  |
| rank | integer | Min Value=0 |  |
| code | String | Not null  Size: 1 and 5 characters |  |
| active | Char | [‘Y’,’y’,’N’,’n’,’1’,’0’] | Allow only one character |
| visible | Char | [‘Y’,’y’,’N’,’n’,’1’,’0’] | Allow only one character |

**Brand (Brand.java)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Constraint** | **Comments** |
| contactid | Integer |  |  |
| email | String | regExp: "^(.+)@(.+)$" |  |
| countrycode | String |  |  |
| countryname | String |  |  |
| datemodified | Timestamp |  |  |
| usermodified | String | Size: 0 and 50 characters |  |
| sms | String |  |  |
| identifier | String |  |  |
| value | String | Size: 0 and 50 characters |  |
| active | Char | [‘Y’,’y’,’N’,’n’,’1’,’0’] | Allow only one character |

**Contact (Contact.java)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Constraint** | **Comments** |
| description | String | Size: 0 and 50 characters |  |
| code | String | Not null  Size: 1 and 5 characters |  |

**Frequency (Frequency.java)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Constraint** | **Comments** |
| contactid | integer |  |  |
| brandcode | String | Size: 1 and 5 characters |  |
| branddesc | String | Size: 0 and 50 characters |  |
| channelcode | String | Size: 1 and 5 characters |  |
| channeldesc | String | Size: 0 and 50 characters |  |
| frequencycode | String | Size: 1 and 5 characters |  |
| frequencydesc | String | Size: 5 and 50 characters |  |
| rank | integer | Min Value=0 |  |

**Brand Communication Channel (BrandChannel.java)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Type** | **Constraint** | **Comments** |
| channelfreqid | integer |  |  |
| channelcode | String | Size: 1 and 5 characters |  |
| channeldesc | String | Size: 0 and 50 characters |  |
| frequencycode | String | Size: 1 and 5 characters |  |
| frequencydesc | String | Size: 5 and 50 characters |  |
| rank | integer | Min Value=0 |  |
| active | Char | [‘Y’,’y’,’N’,’n’,’1’,’0’] | Allow only one char |
| visible | Char | [‘Y’,’y’,’N’,’n’,’1’,’0’] | Allow only one char |

**Communication Channel Frequency (ChannelFrequency.java)**

|  |  |  |
| --- | --- | --- |
| **Variable** | **Type** | **Comments** |
| code | integer | 18F specific response code |
| description | String |  |
| comments | String |  |

**Response Code (ResponseCode.java)**

# Industry Recommended Available features

This section describes some [optional] general industry recommended available features. These features can be enabled/disabled dynamically via the WebSphere admin web console:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Description** | **Comments** |
| pretty | This is an available query request parameter that causes the web service to output pretty JSON. | ?pretty=true |
| sort | This is an available query request parameter that allows list producing service calls to be sorted based on any available output field. | ?sort=brandcode  ?sort=rank |
| envelope | This is an available query request parameter that instructs the web service to wrap the output in an envelope. | ?envelope=true |

# WAS Global bindings

This section describes the application resource values/constants maintained by WebSphere [environment->naming->name space bindings]:

|  |  |  |  |
| --- | --- | --- | --- |
| **Resource** | **Lookup** | **Value** | **Comments** |
| Authority | epc/authority | aprefwdc01dev01:9080/api/preferencecenter | Used to construct Response URI for create requests. |
| Date Format | epc/date.format | yyyy-MM-dd | Configurable application wide date format. |
| DB Type | epc/db.type | DB | Database type. |
| Envelope | epc/json.envelope.allow | true | Enable envelope feature. |
| Prefix | epc/prefix | /preferencecenter | Used to construct Response URI for create requests. |
| Pretty | epc/json.pretty.allow | true | Enable pretty JSON feature. |
| Scheme | epc/scheme | http | Used to construct Response URI for create requests:[http/https] |
| Sort | epc/json.sort.allow | true | Enable sort feature. |
|  |  |  |  |

# Port Numbers

This section describes the firewall security port numbers that will be used.

|  |  |  |
| --- | --- | --- |
| **Port** | **Protocol** | **Comments** |
| 22 | ssh | Secure socket |
| 80 | http | Web |
| 443 | https | Secure Web |

### Appendix C

### Administrative Resources

This section describes the methods that should be considered as administrative. For example, a normal user or customer should not be able to define new brands, communication channels, etc.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item #** | **Resource** | **Method** | **URI** | **Request Body** | **Response Body** | **Response Header** | **Purpose** |
| 1 | Brand | PUT | /brands/{code} | {  "brand": {  "description": "Flowers",  "rank": 1,  "active": "Y",  "visible": "Y"  }  } | None | 201 Created  Location: [new uri] | Creates a new Brand |
| 2 | Brand | DELETE | /brands/{code} | None | None | 204 No Content | Marks a brand as inactive. |
| 3 | Brand | GET | /brands | None | {  "brands": [{  "brand": {  "code": "FLWS",  "description": "Flowers",  "rank": 1,  "active": "Y",  "visible": "Y"  },  "brand": {  "code": "FAN",  "description": "Fannie Mae",  "rank": 2,  "active": "Y",  "visible": "Y"  }  }]  } | 200 OK | Retrieve a list of brands |
| 4 | Brand | GET | /brands/{code} | None | {  "brand": {  "code": "FLWS",  "description": "Flowers",  "rank": 1,  "active": "Y",  "visible": "Y"  }} | 200 OK | Retrieves a brand |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item #** | **Resource** | **Method** | **URI** | **Request Body** | **Response Body** | **Response Header** | **Purpose** |
| 5 | Brand | POST | /brands/{code} | {  "brand": {  “code”: “FLWS”,  "description": "Flowers",  "rank": 1,  "active": "N",  "visible": "Y"}  } | None | 204 No Content | Updates a brand |
| 6 | Communication Channel | PUT | /communicationchannels/ {code} | {  "communicationchannel": {  "description": "email",  "rank": 1,  "active": "Y",  "visible": "Y"  }  } | None | 201 Created [new uri] | Create a new communication channel |
| 7 | Communication Channel | DELETE | /communicationchannels/ {code} | None | None | 204 No Content | Mark a communication channel, inactive |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item #** | **Resource** | **Method** | **URI** | **Request Body** | **Response Body** | **Response Header** | **Purpose** |
| 8 | Communication Channel | GET | /communicationchannels | None | {  "communicationchannels": [{  "communicationchannel": {  "code": "EM",  "description": "email",  "rank": 1,  "active": "Y",  "visible": "Y"  },  “communicationchannel": {  "code": "SMS",  "description": "SMS Text Message",  "rank": 1,  "active": "Y",  "visible": "Y"  }  }]  } | 200 OK | Retrieve a list of communication channels |
| 9 | Communication Channel | GET | /communicationchannels/{code} | None | } "communicationchannel": {  "code": "EM",  "description": "email",  "rank": 1,  "active": "Y",  "visible": "Y"  }} | 200 OK | Retrieve a communication channel |
| 10 | Communication Channel | POST | /communicationchannels/ {code} | {  "communicationchannel": {  “code”: “EM”,  "description": "email",  "rank": 1,  "active": "Y",  "visible": "Y"  }  } | None | 204 No Content | Update an existing communication channel |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item #** | **Resource** | **Method** | **URI** | **Request Body** | **Response Body** | **Response Header** | **Purpose** |
| 11 | Frequency | PUT | /frequencies/{code} | {  “frequency”:  {  “description” : “monthly”  }} | None | 201 Created  Location: [new uri] | Create a new frequency |
| 12 | Frequency | DELETE | /frequencies/{code} | None | None | 204 No Content | Mark a frequency as inactive |
| 13 | Frequency | GET | /frequencies | None | {  "frequencies": [{  "frequency": {  "code": "MON",  "description": "Monthly"  },  "frequency": {  "code": "WEEK",  "description": "Weekly"  }  }]  } | 200 OK | Retrieve a list of frequencies |
| 14 | Frequency | GET | /frequencies/{code} | None | {  "frequency": {  "code": "MON",  "description": "Monthly"  }  } | 200 OK | Retrieve a frequency |
| 15 | Frequency | POST | /frequencies/{code} | {  “frequency”:  {  “code”: “MON”,  “description” : “monthly”  }} | None | 204 No Content | Update a frequency |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item #** | **Resource** | **Method** | **URI** | **Request Body** | **Response Body** | **Response Header** | **Purpose** |
| 16 | Communication Channel Frequency | PUT | /communicationchannelfrequencies | {  “communicationchannelcode”: “EM”,  “frequencycode”: “MN”,  “rank”: 3,  “active”: “Y”,  “visible” : “Y”  }  } | None | 201 Created  Location:[new uri] | Create a communication channel frequency |
| 17 | Communication Channel Frequency | DELETE | /communicationchannelfrequencies/{id} | None | None | 204 No Content | Mark a communication channel frequency as inactive |
| 18 | Communication Channel Frequency | GET | /communicationchannelfrequencies/{id} | None | {  "communicationchannelfrequencies": [{  " communicationchannelfrequency ": {  “id”: 2,  “communicationchannelcode”: “EM”,  “communicationchanneldesc”: “email”,  “frequencycode”: “MN”,  “frequencydesc”: “Monthly”  “rank”: 3,  “active”: “Y”,  “visible” : “Y”  }  }]  } | 200 OK | Get a list of all communication channel frequencies |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item #** | **Resource** | **Method** | **URI** | **Request Body** | **Response Body** | **Response Header** | **Purpose** |
| 19 | Communication Channel Frequency | GET | /communicationchannelfrequencies/{id} | None | {  "communicationchannelfrequency": {  “id”: 2,  “communicationchannelcode”: “EM”,  “communicationchanneldesc”: “email”,  “frequencycode”: “MN”,  “frequencydesc”: “Monthly”  “rank”: 3,  “active”: “Y”,  “visible” : “Y”  }  } | 200 OK | Get a communication channel frequency |
| 20 | Communication Channel Frequency | POST | /communicationchannelfrequencies/{id} | {  “id”: 2,  “communicationchannelcode”: “EM”,  “frequencycode”: “MN”,  “rank”: 3,  “active”: “Y”,  “visible” : “Y”  }  } | NA | 204 No Content | Update a communication channel frequency |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item #** | **Resource** | **Method** | **URI** | **Request Body** | **Response Body** | **Response Header** | **Purpose** |
| 21 | Contact | PUT | /contacts/{contactId} | {  "contact": {  “usermodified” : “system”,  “countrycode” : “USA”,  “email” : “[joe.smo@email.com](mailto:joe.smo@email.com)”  }  } | None | 201 Created  [new uri] | Create a new contact |
| 22 | Contact | DELETE | /contacts/{contactId} | None | None | 204 No Content | Mark a contact as inactive |
| 23 | Contact | GET | /contacts/{contactId} | None | {  "contact": {  “contactid”: 125368745,  “countrycode” : “USA”,  “countryname” : “United States”,  "datemodified": "12/01/15",  "usermodified": "system",  "active": "Y",  “email” : “[joe.smo@email.com](mailto:joe.smo@email.com)”,  “sms” : “440-213-1234”  }  } | 200 OK | Retrieve a contact base on the unique id |
| 24 | Contact | POST | /contacts/{contactId} | {  "contact": {  “usermodified” : “system”,  “countrycode” : “USA”,  “email” : “[joe.smo@email.com](mailto:joe.smo@email.com)”,  “sms” : “440-213-1234”  }  } | None | 204 No Content | Update a contact |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item #** | **Resource** | **Method** | **URI** | **Request Body** | **Response Body** | **Response Header** | **Purpose** |
| 25 | Brand Communication | PUT | /contacts/{id}/communicationchannels/{code}/brands/{code} | {  " brandcommunication": {  "frequencycode": "MON",  “usermodified” : “system”  }  } | NA | 201 Created  Location: [new uri] | Create a brand communication. The value depends on the type of communication. Ex. If SMS then value is the phone number, if email then the value is the email address. Assumption is that the data has already been validated by the client. |
| 26 | Brand Communication | GET | /contacts/{id}/communicationchannels/brands/{code} | None | {  " brandcommunications": [{  " brandcommunication": {  “contacted” : 1927839,  "channelcode": "EM",  "channeldesc": "email",  "brandcode": "FLWS",  "branddesc": "1800flowers",  “frequencycode": "MN",  "frequencydesc": "Monthly"  },  " brandcommunication": {  “contacted” : 1927839,  "channelcode": "EM",  "channeldesc": "email",  "brandcode": "FLWS",  "branddesc": "1800flowers",  “frequencycode": "WK",  "frequencydesc": "Weekly"  } }]} | 200 OK | Get a specific brand communications for a contact |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item #** | **Resource** | **Method** | **URI** | **Request Body** | **Response Body** | **Response Header** | **Purpose** |
| 27 | Brand Communication | GET | /contacts/{id}/communicationchannels/{code}/brands/{code} | None | {  " brandcommunication": {  “contacted” : 1927839,  "channelcode": "EM",  "channeldesc": "email",  "brandcode": "FLWS",  "branddesc": "1800flowers",  “frequencycode": "MN",  "frequencydesc": "Monthly"  }  } | 200 OK | Get a brand communication |
| 28 | Brand Communication | DELETE | /contacts/{id}/communicationchannels/{code}/brands/{code} | None | None | 204 No Content | Mark a brand communication as inactive |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 29 | Brand Communication | POST | /contacts/{id}/communicationchannels/{code}/brands/{code}/brandcommunications/{id} | {  " brandcommunication": {  "frequencycode": "MON",  “usermodified” : “system”  }  } | N/A | 204 No Content | Update a brand communication |
| 30 | Brand Communication | GET | /contacts/{id}/communicationchannels/brands | None | {  " brandcommunications": [{  "brandcommunication": {  “contacted” : 1927839,  "channelcode": "EM",  "channeldesc": "email",  "brandcode": "FLWS",  "branddesc": "1800flowers",  “frequencycode": "MN",  "frequencydesc": "Monthly"  },  " brandcommunication": {  “contacted” : 1927839,  "channelcode": "CTLG",  "channeldesc": "catalog",  "brandcode": "FLWS",  "branddesc": "1800flowers",  “frequencycode": "MN",  "frequencydesc": "Monthly"  }  }]  }  }]  } | 200 OK | Get all brand communications. |

|  |  |  |
| --- | --- | --- |
| **Date** | **Changes\_by** | **Description** |
| 2/29/2016 | Casmon Gordon | Initial Version |
| 2/29/2016 | Casmon Gordon | Modified document so that PUT is used for create requests and POST for all update requests. |
| 2/29/2016 | Casmon Gordon | Changed rank values to be integer. (removed quotes around values) |
| 2/29/2016 | Casmon Gordon | Changed active and visible values to be either “Y” or “N”. Removed references to integer 1. |
| 2/29/2016 | Casmon Gordon | For communication channel get all request remove {id} from URI as this is not needed. (item 18) |
| 2/29/2016 | Casmon Gordon | Added email and value to contact requests/responses and also contact logical object (per discussion with Ian Kogen) |
| 4/25/2016 | Casmon Gordon | Updated  Response codes  Java version  API libraries used  Design Patterns  Log4J2 specifics  Class Design - Contact added countrycode, countryname, email, sms  Service Call Description  Added  Port number section  Security section  JSON Responses section  Industry Recommended Available Features |
|  |  |  |
|  |  |  |

## Change Log