BRITEVERIFY

Developer Guide

Subject:	**BriteVerify Developer Guide**
Author:	**James McLachlan**
Modified Date:	**October 27, 2011**
Revision Number:	**1.1**
Sensitivity:	**Public**

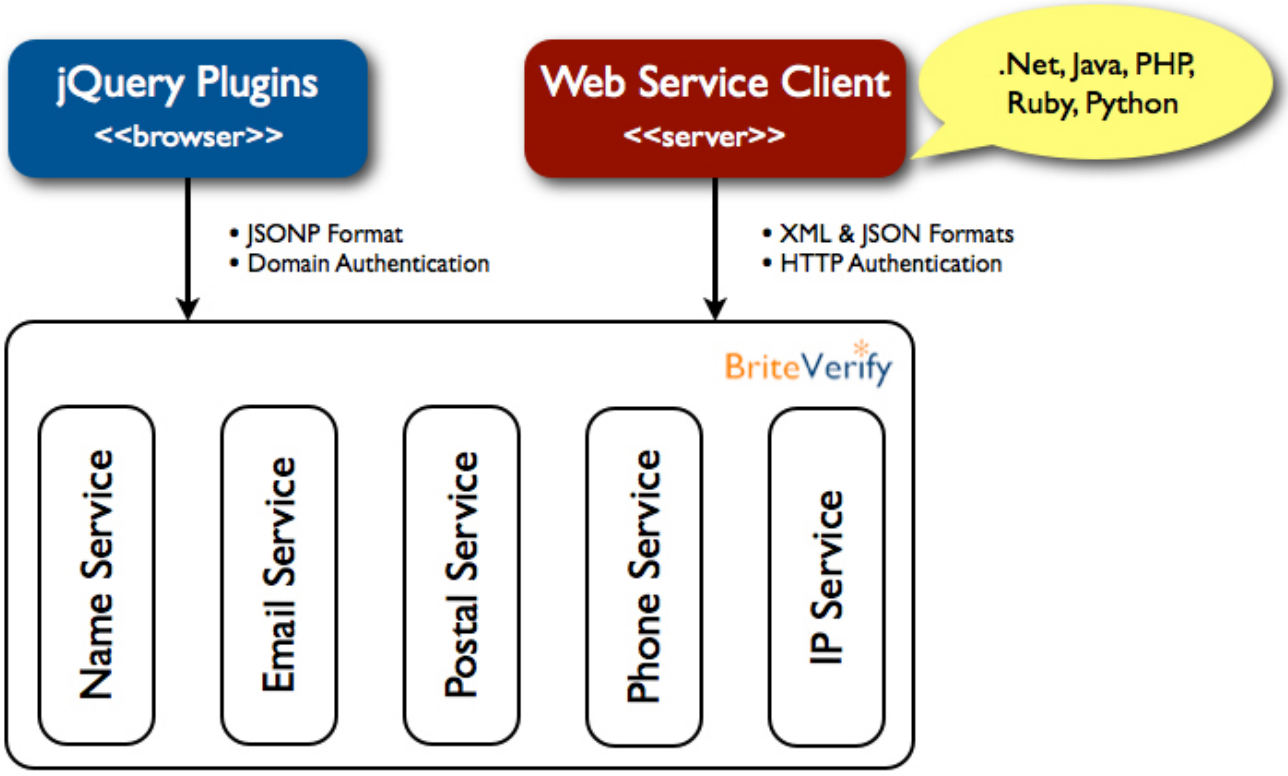
This document provides technical documentation and examples of the BriteVerify Platform. It is intended for developers and the technically minded. A basic grasp on JavaScript, jQuery, HTML and modern Web programming languages is strongly suggested.

BriteVerify

Table of Contents

DEVELOPING WITH BRITEVERIFY

Overview



BriteVerify is a suite of RESTful Web Services and JavaScript APIs for the real-time verification of consumer information. The data elements available for verification are:

- Name
- Email
- Postal Address
- Phone
- IP Address

The JavaScript API is designed to be used inside the browser and require no backend server integration. Simply cut and past a code snippet into the body of any HTML page with a form and you are ready to BriteVerify. For developers who wish to integrate on the backend, BriteVerify RESTful Web Services can be consumed with nothing more than an HTTP client and an XML or JSON parser in almost any language or platform (.NET, Perl, PHP, Ruby, Java...).

Types Verification

Name

The name verification service applies profanity filters to the name then attempts to identify gender.

Email

The email service provides 3 escalating steps to ensure that a consumer's email is correct and deliverable.

1. Basic syntax and format validation
2. Domain and MX Record checks (to ensure the domain is real, active, and capable of receiving email)
3. Account Verification (ensures that the given account is active on the domain)

In the third step, account verification, BriteVerify opens an SMTP conversation with the email server and attempts to validate that the account is active and capable or receiving email. The service can verify email account on nearly every major provider.

Postal Address

The default behavior of the Postal Address verification is to ensure not only that the address is fully coded but also that it is USPS Delivery Point Verified, or DPV. This means that the address is not simple correct but that it is capable of receiving mail from the US Postal Service.

Phone

By default, the standard phone service applies a simple area code + prefix check against a database of known area code and prefix matches. Premium phone verification ensures that the phone number is a dialable number and is capable of filtering based on phone type (mobile, landline, digital, directory assistance, toll free, etc.).

NOTE: We can only ensure that a number is dialable, not that is necessarily connected and working. Dialable means that the North American Numbering Plan (NNAP) has assigned the number to a service provider. This does not mean that the number is not out of service. There is no way to ensure that a number has not been disconnected other than physically calling that number.

BriteVerify Authorization & Authentication

There are two methods of authorization for BriteVerify that depend on the type of client employed. The JavaScript API uses domain-based authorization, while the standard Web Service clients use traditional basic HTTP authentication. Both of these methods are available over HTTPS.

Domain-Based Authorization

Since the JavaScript API calls execute directly in the browser traditional basic HTTP authentication is not an option as the username and password are easily exposed. To get around this issue BriteVerify references each request against a user supplied list of authorized domains. If the request comes from a domain not on the list, BriteVerify responds with an error and does nothing. During testing a certain number of test transactions can come from unauthorized domains before BriteVerify starts rejecting requests.

API Key Authentication

Once your account is active, each request from your servers should include your apikey as shown below:

```
http://api.briteverify.com/emails/verify.xml?email[address]=jdoe@briteverify.com&apikey=your-apikey-here
```

Authorization Errors

If no valid form of authentication is provided then BriteVerify will despond with an unauthorized error message, and a HTTP status code of 401.

```
<?xml version="1.0" encoding="UTF-8"?>
<errors>
   <error>User not authorized</error>
</errors>
```

How to BriteVerify

There are two different ways to get up and running with BriteVerify :

1. BriteVerify JavaScript API
2. Web Service integration

BriteVerify JavaScript API

The BriteVerify JavaScript API sits atop jQuery and the functions follow a similar pattern to traditional jQuery AJAX calls in that an optional callback function may be passed to that will execute when the call completes. All BriteVerify functions are contained within the briteVerify namespace to avoid potential naming conflicts. All the functions follow a similar pattern for their method signature: value, callback. The first parameter is the value to be verified, either a simple string for functions such as email, or a JSON object, as with address. The second parameter it the callback function to be executed when the verification is complete. This function will always have the resulting object passed into it. This is illustrated in the following code example:

```
jQuery.briteVerify.email("myemail@somedomain.com", function(email){
```

```
    alert(email.address); // myemail@somedomain.com
}
```

Handling Errors

Errors Array

If the value cannot be verified, the response object will contain an array of errors. The errors array is an array of arrays. Each individual error is an array where the first value is the name of the attribute on which the error occurred and the second is the error message itself. Almost always there is only one error per field. So checking for additional errors is usually pointless. However, it can be valuable to know to which part of the value the error applies. For example, with the email response object contains the address, domain, and account attributes and the error will apply to one of those.

```
jQuery.briteVerify.email("myemail@somedomain.com", function(email){
    // first check to see if there are any errors
    if(email.errors){
            // get the first error in the array
            var error = email.errors[0];
            var attr = error[0];
            var msg = error[1];
            // so a different error message base on the attr
            if(attr == "address"){
                    alert("incorrect email syntax");
            }else if(attr == "account"){
                    alert("the account was not found on the domain");
            }else if(attr == "domain"){
                    alert("the is not valid");
            }

    }
}
```

Errors JSON Object

In addition to the basic errors array, there is also an errors_json object that is essential a utility object for the errors array. It makes coding a little easier and cleaner.

```
jQuery.briteVerify.email("myemail@somedomain.com", function(email){
    // first check to see if there are any errors
    if(email.errors_json){
            // so a different error message base on the attr
            if(email.errors_json.address){
                    alert("incorrect email syntax");
            }else if(email.errors_json.account){
                    alert("the account was not found on the domain");
            }else if(email.errors_json.domain){
                    alert("the is not valid");
            }
    }
```

```
}
```

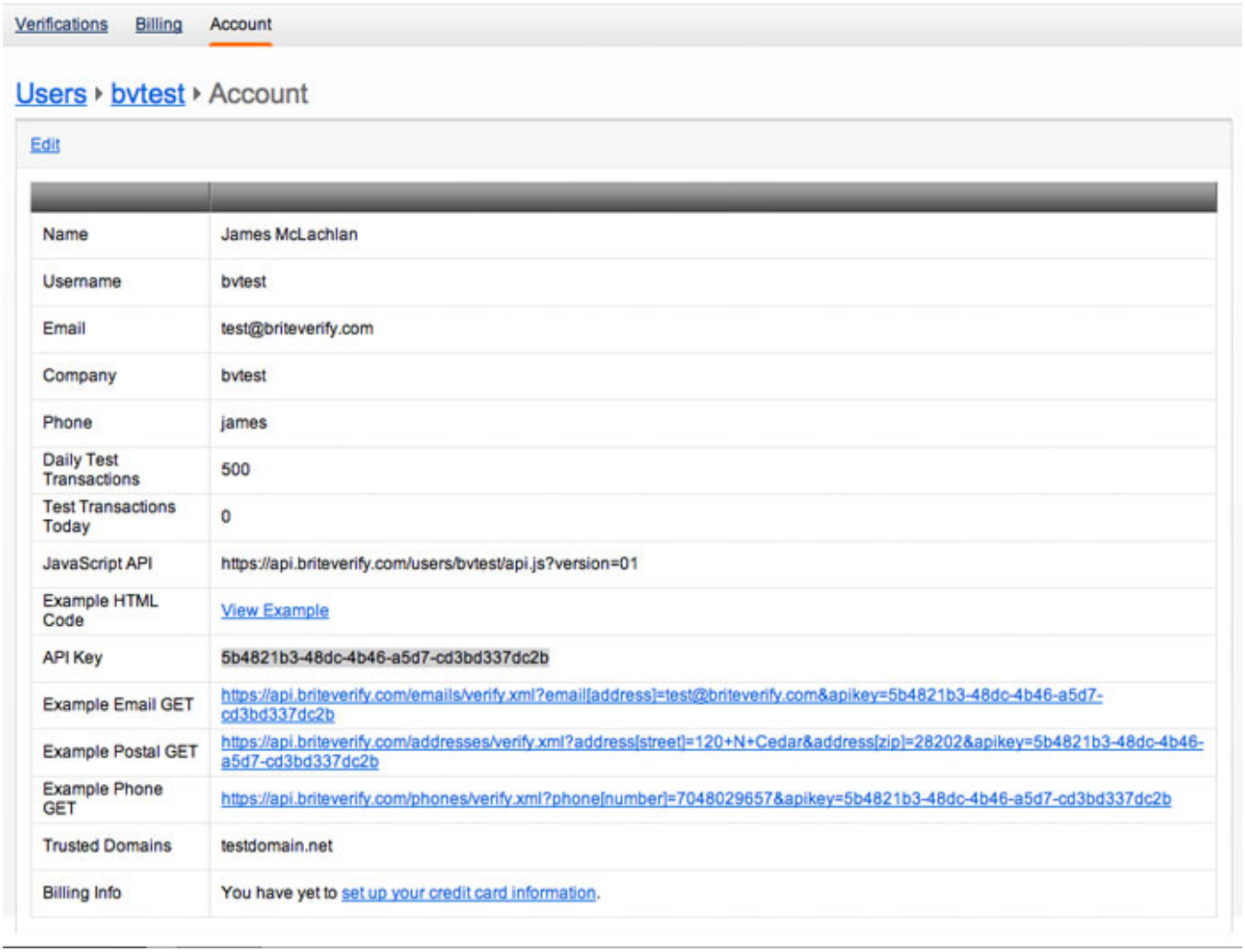
User Account Errors

Should your account surpass its daily test transaction limit for unauthorized domains, or if your account balance reaches 0, an error will be returned on the "user" attribute. It is a best practice to ignore these messages, but they can be helpful for testing purposes. A notification will be sent out prior to any user account hitting a zero balance, or if auto-billing is enabled a billing notification will be sent.

```
jQuery.briteVerify.email("myemail@somedomain.com", function(email){
    // first check to see if there are any errors
    if(email.errors_json){
           // so a different error message base on the attr
           if(email.errors_json.user){
                  alert("user account: " + email.errors_json.user); //"overbalance"
           }
    }
}
```

BriteVerify Web Services API

The BriteVerify Web Services API is a simple HTTP GET request that responds in either XML or JSON based on the format you request. Each GET request must include an apikey paramater in addition to the data to be verified. To obtain you secret API key, log into BriteVerify and click the "account" link.



Using the verify GET URL you can explore the resulting xml of various calls. It does follow the Rails field naming convention, which is "object[attribute]," as illustrated in the examples below.

Verify GET Exampled

```
https://api.briteverify.com/emails/verify.xml?email[address]=jdoe@briteverify.com&apikey=your-apikey-here
```

```
https://api.briteverify.com/emails/verify.js?email[address]=jdoe@briteverify.com&apikey=your-apikey-here
```

Result

Both calls should respond with an HTTP status code of "201 Created" and the following xml response document.

```
<?xml version="1.0" encoding="UTF-8"?>
<email>
    <address>jdoe@briteverify.com</address>
    <domain>briteverify.com</domain>
    <account>jdoe</account>
</email>

or in JSON using verify.js

{"email":
    {"address":"jdoe@briteverify.com",
    "value":"jdoe@briteverify.com",
    "domain":"briteverify.com",
    "account":"jdoe",
    "status":"valid",
    "duration":0.002432}
}
```

If, however, the email is invalid the HTTP status code will be a 422 (Unprocessable Entity) and the XML or JSON payload will contain the errors.

```
<?xml version="1.0" encoding="UTF-8"?>
<errors>
    <error>Account jerk not found at briteverify.com</error>
</errors>

or in JSON

{"email":
    {"address":"jdoe@briteverify.com",
    "value":"jdoe@briteverify.com",
    "domain":"briteverify.com",
    "account":"jdoe",
    "errors":[["account","jdoe not found at briteverify.com"]],
    "status":"not_found",
    "duration":0.002432}
}
```

PLEASE NOTE HTTP 422 Status Code

In many Web Development frameworks, such as .NET, a 422 status code will raise an exception. This is the appropriate behavior, as it is an exceptional case. 400 level status codes indicate an application level exception, in this case, that an email is invalid. If you are not familiar with REST

based services and are a .NET developer a nice intro is provided by Yahoo at

http://developer.yahoo.com/dotnet/howto-rest_vb.html

Authorization Errors

If no valid form of authentication is provided then BriteVerify will despond with an unauthorized error message, and a HTTP status code of 401.

```
<?xml version="1.0" encoding="UTF-8"?>
<errors>
   <error>User not authorized</error>
</errors>

or in JSON

{"error":"User not authorized"}
```

BriverVerify API

Name

jQuery.briteVerify.name(name, callbackFunction)

Accepts the full name of a consumer and a callback function. A name object is passed into the callback function with the results of the call.

```
name.first_name;
name.middle;
name.last_name;
name.prefix;
name.suffix;
name.gender;
name.errors; // an errors array with all errors returned by the service
name.errors_json; // an errors JSON object with all errors returned by the service
```

Example

```
jQuery("#myNameFieldId").change(function(){
  // in this example there is just one name field into which the user types there
full name
  // the callback function should have the name object passed back into it.
  jQuery.briteVerify.name(this.value, function(name){
    if(name.errors){
      alert("Your name is not valid. Please correct");
    }else{
```

```
        var fullname = name.first_name + " " + name.middle + " " + name.last_name;
        alert(fullname + " is a valid name");
      }
    });
});
```

Name Web Service

GET Example

```
http://api.briteverify.com/names/verify.xml?name[fullname]=james+mclachlan&apikey=y
our-apikey-here
```

Response

```
Status: 201 Created
<?xml version="1.0" encoding="UTF-8"?>
<name>
  <first-name>James</first-name>
  <fullname>James McLachlan</fullname>
  <gender>male</gender>
  <last-name>McLachlan</last-name>
  <middle-name> </middle-name>
</name>
```

Parameters

- name[fullname]
- name[first_name]
- name[last_name]
- name[middle_name]

Email

The email service is slightly different from the other services in that it returns a status for emails that appear to be valid but cannot be entirely verified at the account level. If an email address is syntactically correct and the domain is capable of receiving email, but the existence of the inbox cannot be verified either way then the status of the email is treated the same as a valid address but the status is set to “unknown.” All verifications with a status of “unknown” are not billed.

jQuery.briteVerify.email(email, callbackFunction)

Accepts an email string and a callback function. An email object is passed into the callback function, which is returned when processing is complete:

```
email.address; // jdoe@nodo.com
email.domain;  // nodo.com
```

```
email.account; // jdoe
email.status;  // "valid" or "unknown"
email.errors;  // [["domain", "does not exist"]]
email.errors_json;  // {account : "not found on nodo.com"}
```

Example

```
jQuery("#myEmailFieldId").change(function(){
  jQuery.briteVerify.email(this.value, function(email){
    if(email.errors){
      if(email.errors_json.domain){
        alert("Your email domain " + email.errors_json.domain);
      }else if(email.errors_json.address){
        alert("Your email " + email.errors_json.address);
      }else if(email.errors_json.account){
        alert("Your account " + email.errors_json.account);
      }
    }else if(email.status == "unknown"){
        alert("Email status is unknown");
    }
  });
});
```

Email Web Service

GET Example

```
http://api.briteverify.com/emails/verify.xml?email[address]=jdoe@briteverify.com&ap
ikey=your-apikey-here
```

Response

```
Status: 201 Created
<?xml version="1.0" encoding="UTF-8"?>
<email>
  <address>jdoe@briteverify.com</address>
  <domain>briteverify.com</domain>
  <account>jdoe</account>
  <status type="symbol">valid</status>
</email>
```

Parameters

- email[address]

Address

jQuery.briteVerify.address(address, callbackFunction)

Accepts an address JSON object and a callback function. An address object is passed into the callback function, which is returned when processing is complete:

```
address.street;
address.unit;
```

```
address.city;
address.state;
address.zip;
address.plus4;
address.address_type;
address.errors;
address.errors_json
```

Example

```
jQuery("#myZipFieldId").change(function(){
  // check first to see if the street field has something in it
  var streetVal = jQuery("#myStreetFieldId").val();
  if(streetVal){
    var myAddress = { street : streetVal, unit : jQuery("#myUnitFieldId").val(),
city : jQuery("#myCityFieldId").val(), state : jQuery("#myStateFieldId").val(), zip
: jQuery("#myZipFieldId").val() }
  };
  jQuery.briteVerify.address(myAddress, function(address){
    if(address.errors){
      alert("Address not found.");
    }
  });
});
```

Address Web Service

GET Example

```
http://api.briteverify.com/addresses/verify.xml?address[street]=120+nowhere&address
[zip]=28202&apikey=your-apikey-here
```

Response

```
Status: 201 Created
<?xml version="1.0" encoding="UTF-8"?>
<address>
  <address-type>Highrise</address-type>
  <city>Charlotte</city>
  <county>Mecklenburg</county>
  <plus4>1292</plus4>
  <state>NC</state>
  <street>120 N Cedar St </street>
  <unit nil="true"></unit>
  <zip>28202</zip>
</address>
```

Parameters

- address[street]
- address[unit]
- address[city]
- address[state]
- address[zip]

Phone

jQuery.briteVerify.phone(phone, callbackFunction, options)

Accepts a phone string and a callback function. A phone object should be passed into the callback function to hold the result returned when processing is complete. Options are optional and there is currently only a premium flag is supported. If the premium option is passed as true, then the service will execute using the premium phone verification service.

```
phone.number;    // 7041231234
phone.areacode;  // 704
phone.prefix;    // 123
phone.suffix;    // 1234
phone.city;      // Charlotte
phone.county;    // Mecklenburg
phone.country;   // US
phone.errors;    // { number : "is not valid" }
```

Example Default Verification

```
jQuery("#myPhoneFieldId").change(function(){
  jQuery.briteVerify.phone(this.value, function(phone){
    if(phone.errors){
      alert("phone is not valid");
    }
  });
});
```

Phone Web Service

GET Examples

```
http://api.briteverify.com/phones/verify.xml?phone[number]=7041231234&apikey=your-apikey-here
```

Response

```
Status: 201 Created
<?xml version="1.0" encoding="UTF-8"?>
<phone>
  <areacode>704</areacode>
  <city>Charlotte</city>
  <ext nil="true"></ext>
  <number>704 525 5526</number>
  <prefix>525</prefix>
  <service-type>unknown</service-type>
  <suffix>5526</suffix>
</phone>
```

Parameters

- phone[number]

- phone[premium] (true or false)

IP Address

IP Address Web Service

The IP Web Service is a simple get request.

GET Example

```
http://api.briteverify.com/ips/verify.xml?ip=98.139.180.149&apikey=your-apikey-here
```

Response

```
<ip>
   <duration type="float">-0.443712</duration>
   <isp>Yahoo! Inc</isp>
   <status type="symbol">valid</status>
   <zip>94089</zip>
   <city>Sunnyvale</city>
   <value>98.139.180.149</value>
   <domain>yahoo.com</domain>
   <longitude>-122.036350</longitude>
   <latitude>37.368830</latitude>
   <region>California</region>
   <country>United States</country>
   <address>98.139.180.149</address>
</ip>
```