

Veriplace Developer Guide

Copyright © 2009 WaveMarket, Inc.

All rights reserved. You cannot print any part of this document without permission from the copyright owner.

WAVEMARKET MAKES NO EXPRESS OR IMPLIED WARRANTIES WITH RESPECT TO THE CONTENT INCLUDED IN DEVELOPER'S GUIDE AND EXPRESSLY DISCLAIMS WARRANTIES OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT, REPRESENTATION OR WARRANTY ARISING BY USAGE OF TRADE, COURSE OF DEALING OR COURSE OF PERFORMANCE. BY ACCESSING THE DEVELOPER'S GUIDE, DEVELOPER ACKNOWLEDGES AND AGREES THAT IT IS NOT ENTITLED TO RELY ON THE ACCURACY OR COMPLETENESS OF THE CONTENT HEREIN.

THIS DEVELOPER'S GUIDE DOES NOT CONSTITUTE OR IMPLY A LICENSE, EXPRESS OR IMPLIED, TO ANY INTELLECTUAL PROPERTY RIGHTS OR A COMMITMENT OR BINDING CONTRACT REGARDING THE CONTENT HEREIN. USE OF THE CONTENT INCLUDED HEREIN IS SUBJECT TO WAVEMARKET'S SEPARATELY STATED DEVELOPER TERMS OF SERVICE.

IN THE EVENT OF ANY CONFLICT AMONG THE TERMS AND CONDITIONS INCLUDED IN THE DEVELOPER TERMS OF SERVICE AND THOSE INCLUDED IN THIS DEVELOPER'S GUIDE, THE TERMS AND CONDITIONS INCLUDED IN THE DEVELOPER TERMS OF SERVICE WILL GOVERN. WAVEMARKET RESERVES THE RIGHT TO MAKE CHANGES TO THIS DEVELOPER GUIDE, IN WHOLE OR PART, FROM TIME TO TIME WITHOUT NOTICE.

WAVEMARKET PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING OR LIFE SUSTAINING APPLICATIONS.

Table of Contents

1. About Veriplace	3
2. Getting Started	9
3. Using the User Discovery API	19
4. Using the Get Location API.....	24
5. Using the Past Location API	38
6. Using the Verify Permission API.....	40
7. Using the Permission Delete API.....	42
8. Using the Get Permissions API	44
9. Using the Invitation API	46
10. Using the Polling API	51
11. Using the Verify Locatability API.....	54

12. Previous API versions56

A. Veriplace XML Schema59

1. About Veriplace

Veriplace is a comprehensive web platform that provides mobile users' location information to 3rd party applications using a standards-based developer API while empowering users to control when and how their locations can be accessed.

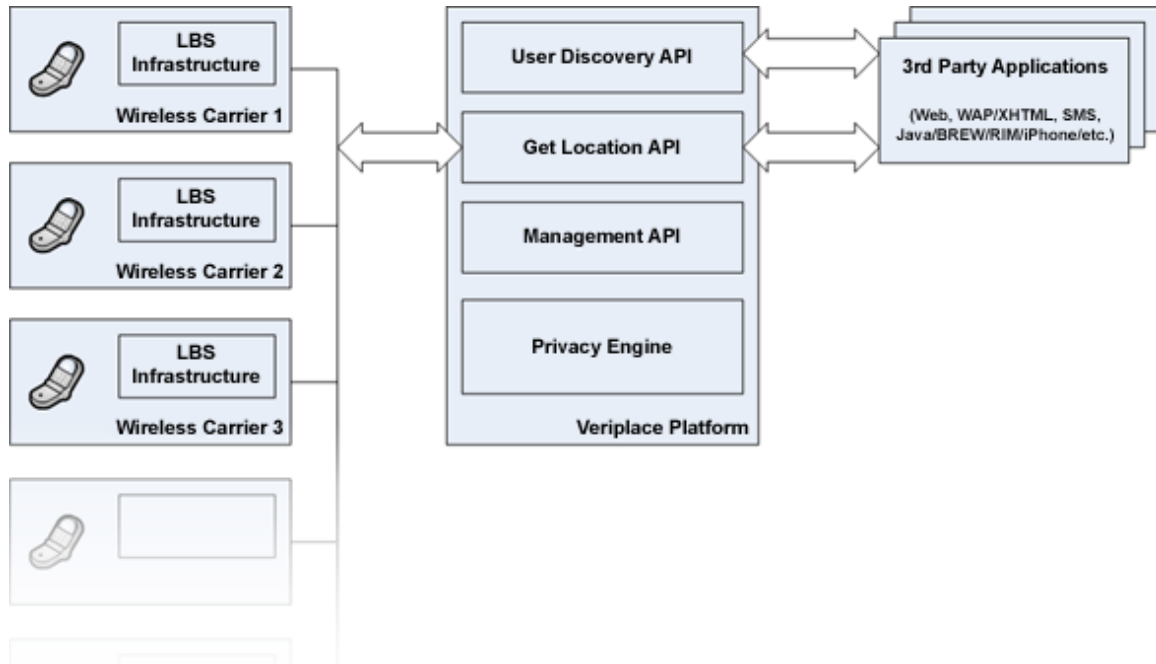
Veriplace is built to access location information from all Wireless Carriers and position determination technologies, including GPS, A-GPS, cell triangulation and cell tower solutions.

Veriplace supports all major mobile application platforms, including Web, WAP (mobile web), SMS, and native client (J2ME, BREW, Smart Phone).

1.1 Veriplace Architecture

From the perspective of an application developer, the Veriplace platform acts as an interchange between Wireless Carrier location data and 3rd party applications. Developer access to location is controlled by the API and by user-configured privacy rules.

[The relationship between Veriplace, Wireless Carriers, 3rd party applications, and the Veriplace API is illustrated in the diagram below.](#)



above: Veriplace architecture from a developer's perspective

Veriplace sits between the wireless carrier network (on the left) and the developer of 3rd party applications (on the right). As a developer for Veriplace, you have direct access to the **User Discovery API** and the **Get Location API** and indirect access to the **Privacy Engine** and to **Wireless Carriers**.

- You use the **User Discovery API** to identify users. For more detailed information, see [3.0 Using the User Discovery API](#) (page 19).
- You use the **Get Location API** to obtain user location information. For more detailed information, see [4.0 Using the Get Location API](#) (page 24).
- The **Privacy Engine** manages user privacy decisions and controls location access using a fine-grained permission model.
- Veriplace manages all communications and integration with **Wireless Carriers**, which provide network-specific position determination technologies.

1.2 Wireless Carriers

Wireless Carriers employ a variety of network technologies to extract an individual user's location information. Since network architectures vary considerably among carriers, one of Veriplace's advantages is that it flexibly manages the requirements for communicating with different carrier technologies. Your application does not need to know or understand how location was obtained.

For example, many wireless devices today are equipped with GPS. In some cases, Wireless Carriers have deployed network-based solutions that allow access to GPS data from such devices; in other cases, similar solutions exist to access cell sector, cell tower, or cell triangulation data. It may also be possible to provision a mobile device with customized software that accesses location data and returns it to the Veriplace platform.

Where these technologies have varying costs or limitations, Veriplace abstracts the notion of subscriber location so developers can focus on what's important and ignore the underlying position determination technology.

Most significantly, deriving the location of a mobile device can incur a cost at the underlying carrier. This cost is passed back to the Veriplace platform and onto the application developer. Veriplace supports a variety of pricing and business models to best support the requirements of the application developer.

1.3 Privacy Engine

Before allowing access to user location data, the Veriplace Privacy Engine authenticates and authorizes each 3rd party application request.

Individual users have authority over which applications can access their location. Veriplace provides users with a wide range of control over their permission preferences and also automates any carrier-specific logic related to permissioning decisions. For example, if your application wants to locate a mobile device that is being used by a minor, some carriers may require the authorization of and/or notification to the mobile account's owner, presumably the parent or custodian for the minor.

Veriplace takes responsibility for identifying and notifying the owner using whatever carrier system is available - your application doesn't have to anticipate this situation and others like it.

1.4 Platform Open Standards

Veriplace uses standards-based technologies, including HTTP, XML and OAuth (see below), which can be employed from any environment that developers prefer. Libraries for these technologies are available on every major development platform.

1.4.1 Advantages of Open Standards

Veriplace employs the OAuth standard in conjunction with XML over HTTP to provide greater power and easier integration when compared to existing location-based service standards, such as MLP and ParlayX. See [2.3.1 Using OAuth](#) (page 10) for more details. As a result, Veriplace gains the following advantages:

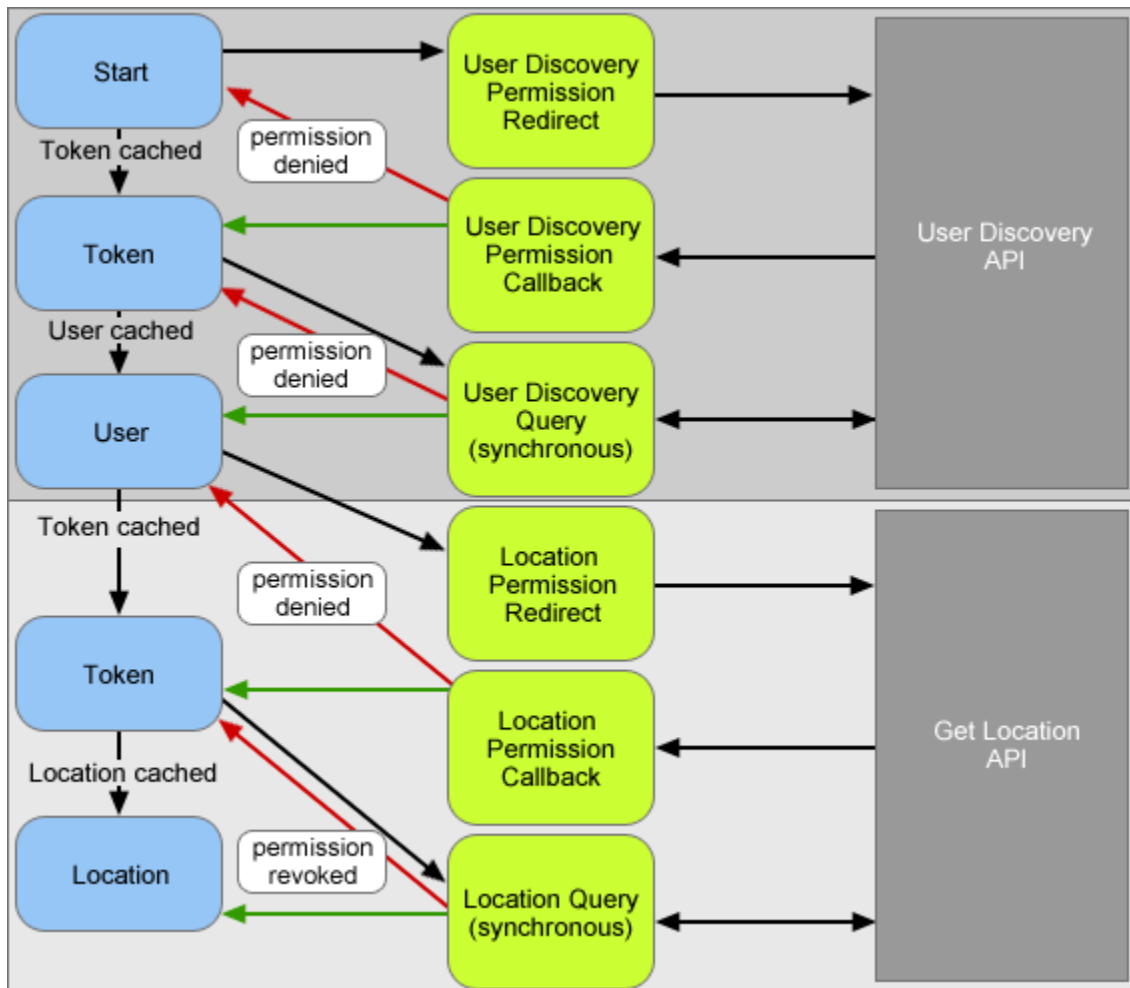
- **Simplicity** - existing standards are overly verbose and subject to many proprietary variations, which can make them difficult to use. Veriplace prefers a simpler and more effective approach, such as employing location modes instead of multiple parameters. See [4.2.6 Location Modes](#) (page 27) for more details.
- **Ease of development** - Veriplace adheres to the principles of REST, the open source architectural style advocating pragmatic practice with industry standards and eschewing complex request encodings in favor of well-defined HTTP URLs. This style permits more rapid development than existing location-based service standards, which often require hand-crafted XML or specialized libraries.
- **Fine-grained privacy** - existing standards support the rejection of location requests for privacy reasons (yes/no), but they provide no means for obtaining permission

from users, nor do they expose nuanced privacy decisions at a finer grain than just yes/no.

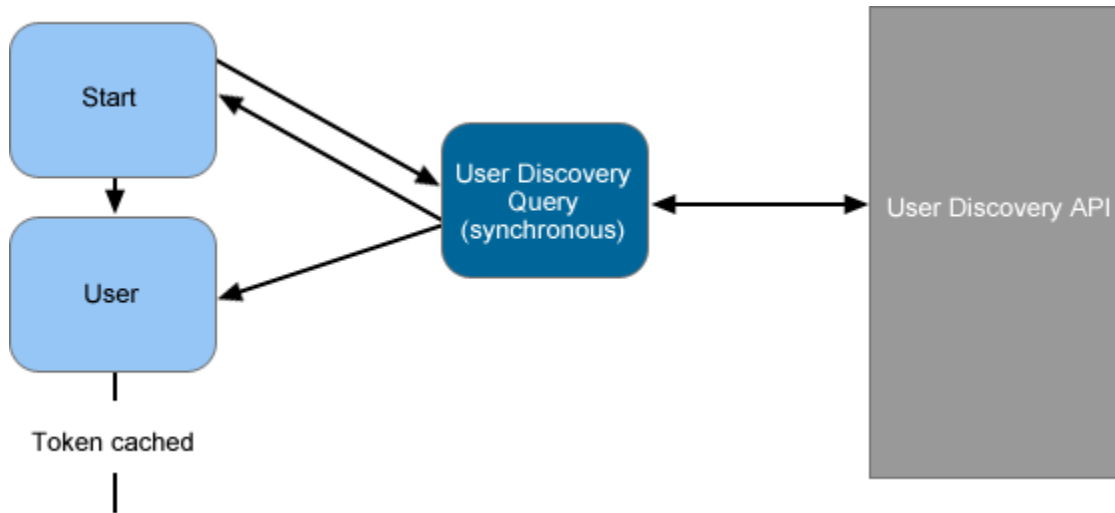
- Feature richness - Veriplace APIs add user discovery, the ability to update location, and other functionality that follows from common application use cases.

1.4 Workflow

The general workflow for the majority of location requests is illustrated below.



There are two sections to the workflow – the top represents the User Discovery API, and the bottom the Location API. The former shows the User Discovery workflow with the user login method. This is described in more detail in [3.0 Using the User Discovery API](#) (page 19). A modified version of the top section is shown below for User Discovery using Personal Identification Information (PII) – see [3.2.1 Applications Supply PII Data](#) (page 19).



The column on the left describes the state needed by your application as you fetch token and location information. You can find more information on this flow in section [2.3.1.6 OAuth for Location Workflow](#) (page 13). The column in the center describes the API events that occur as your application interacts with Veriplace. The column on the right shows how each API is involved during the workflow process.

1.6 API Overview

Veriplace developers have access to several different APIs. Most developers will only need to use two of these:

- **User Discovery API** lets developers discover user identities. See [3.0 Using the User Discovery API](#) (page 19).
- **Get Location API** lets developers obtain individual user location information. See [4.0 Using the Get Location API](#) (page 24).

Additionally, some developers may find the following APIs useful:

- **Past Location API** lets developers re-acquire individual user location information that was previously obtained. See [5.0 Using the Past Location API](#) (page 38).
- **Verify Permission API** queries the server directly (rather than using interactive user authorization) for an existing permission to locate a user. See [6.0 Using the Verify Permission API](#) (page 40).
- **Permission Deletion API** lets developers surrender permission to access individual user location information. See [7.0 Using the Permission Delete API](#) (page 42).

1.7 Other Documentation

See the following documents for more information:

- **Veriplace Developer Content Requirements** for details about what types of applications and application behaviors are not allowed on the Veriplace platform.

- **Veriplace SDK** for details about downloading, installing and using the Veriplace SDK materials.

2. Getting Started

To develop applications for Veriplace, you will need to:

- Register with Veriplace.
- Understand how your application is allowed to use location data.
- Integrate with the Veriplace API.
- Review the path to certification.

2.1 Registering with Veriplace

You must register with Veriplace before you can start developing against its APIs.

To register, follow these steps:

1. **Register for a developer account on the Veriplace Developer Portal:**
<http://developer.veriplace.com>¹
As part of this process, you will need to provide an email address and agree to the Developer Terms of Service.
2. **Sign in to the developer site to create and manage your applications.**
Each developer application receives credentials for using the APIs. For more details, see section [2.3.1.2 OAuth Credentials](#) (page 11).
3. **Build your application.**
Your application can use the Veriplace SDK to get started quickly.
4. **Test your application using simulated and/or developer-enabled accounts.**
Applications in development are not allowed to locate the majority of Veriplace users; however each developer is assigned several simulated user accounts that may be used for testing purposes. In addition, non-simulated accounts may be developer-enabled to test with real location.

Once you are confident your application performs the way you want it to, you can submit it to Veriplace for testing and review. For more details, see section [2.4 Certifying and Launching Your Application](#) (page 17).

2.2 Using Location Data

The Veriplace Developer Terms of Service places the following restrictions on how location data you obtain using Veriplace may be used:

- Developers may not resell, aggregate, or use the data for any purpose other than what was specifically agreed to during developer registration or by a subsequent agreement.
- Developers may not retain or employ location data after its expiration date. The expiration date is specified in the Veriplace XML schema document. For more information, see [4.2.7 Expiration Dates](#) (page 28).
- There is no need for developers to store location data themselves. Every location returned by Veriplace includes an identifier that can be used to subsequently retrieve it using the Past Location API. For more information, see section [5.0 Using the Past Location API](#) (page 38).

Developers wanting to store location data must:

- Ensure that personally identifiable information data (PII data) are not associated with position data in a non-secure form. Typically, encrypting the position data is sufficient.
- Delete location data within 24 hours of its expiration.

If you have any questions about use or misuse of location data, contact your Veriplace representative at WaveMarket.

Note: WaveMarket subscribes to the [CTIA Best Practices and Guidelines for Location Based Services](#)².

2.3 Understanding the Veriplace APIs

To understand the Veriplace APIs, it is essential to understand the **OAuth** protocol and how it secures **Protected Resources**

2.3.1 OAuth

For security and privacy reasons, Veriplace restricts developer access to user identity and user location. These are considered protected resources, and typically require *permission* from the user before they can be accessed.

The Veriplace Developer API uses the OAuth standard for its API requests, which provides a framework for obtaining permission to access such protected resources. OAuth supplements standard HTTP requests, allowing XML encodings of these resources to be returned if permission is granted. While the OAuth protocol may be novel for some developers, library implementations are available for most development environments.

Note: We assume you are familiar with HTTP and XML in your preferred development environment.

The OAuth interaction leverages HTTP redirection to shift the responsibility for authorization away from 3rd party applications onto Veriplace. This model allows Veriplace to be the only site that must be trusted for privacy decisions (by users and Wireless Carriers) and relieves 3rd party applications of this burden. To minimize the usability impact of HTTP redirection, Veriplace possesses co-branding facilities to enforce consistency between your application and the Veriplace UI.

2.3.1.1 OAuth Request URLs

The OAuth specification defines three request URLs, which Veriplace implements as:

- **Request Token URL:**

```
https://veriplace.com/api/requestToken
```

- **User Authorization URL:**

```
http://veriplace.com/api/userAuthorization
```

- **Access Token URL:**

```
https://veriplace.com/api/accessToken
```

With the exception of User Authorization redirection, Veriplace APIs require HTTPS.

Requests for Veriplace's User Authorization URL uses one required and one optional parameter: *uri* and *immediate*. You can find more information on the parameters in section [2.3.1.5 OAuth for User Discovery Workflow](#) (page 12).

2.3.1.2 OAuth Credentials

Before using the Veriplace APIs, you must obtain the following:

- **An OAuth consumer key** used for signing requests
- **An OAuth consumer secret** use for signing request
- **An application-specific OAuth Access Token** used to obtain certain protected resources

Your application's credentials are readily available from within the **Veriplace Developer Portal**.

2.3.1.3 More Information

Developers unfamiliar with OAuth are encouraged to review the OAuth specification and read one of the online tutorials.

- General information about OAuth can be found at <http://oauth.net>³.

- More specific information about the OAuth specification can be found at <http://oauth.net/documentation/spec>.
- The complete OAuth Core 1.0 specification can be found at <http://oauth.net/core/1.0/>.

Developers can find resources related to using OAuth in several places:

- OAuth code documentation <http://oauth.net/code>
- OAuth code repository <http://oauth.googlecode.com/svn/code/>
- Google Code <http://code.google.com/p/oauth/>

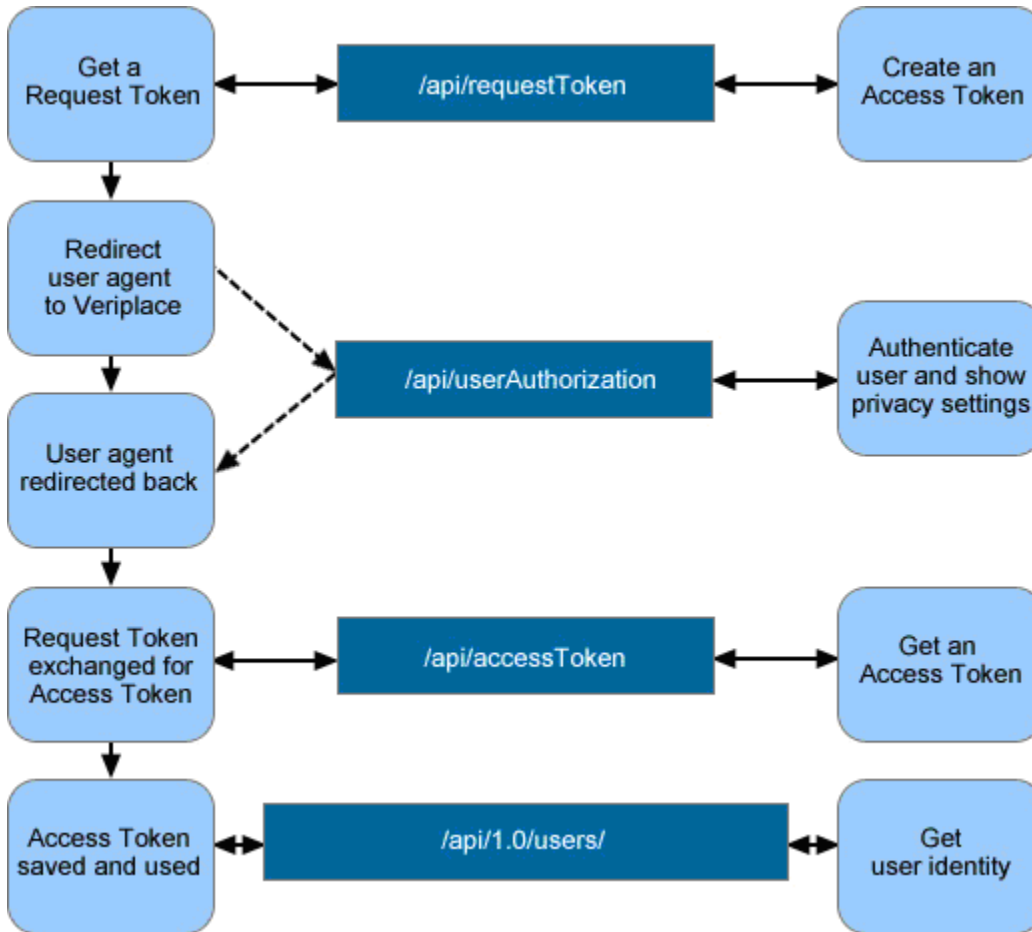
2.3.1.4 How OAuth Works

Veriplace employs OAuth⁴ using the following general workflow:

1. Your application gets a one-time Request Token, which will subsequently be used to retrieve an Access Token if the user grants permission.
2. Your application redirects the User Agent to Veriplace, where the user is asked to provide authorization. The user may answer affirmatively, negatively, or may have the ability to give some form of partial consent. Veriplace subsequently redirects the User Agent back to your application.
3. Your application retrieves an Access Token using the Request Token as a key. Access Tokens correspond to the user's authorization decisions and will only exist if the user provided authorization. If the user did not permit access, no Access Token may be retrieved, and the process ends here.
4. If an Access Token is retrieved, it is used to gain access to the user's location or to some other protected resource.
5. If the Access Token is for a single use, the token is invalidated after use; otherwise the Access Token can be re-used and may be cached.

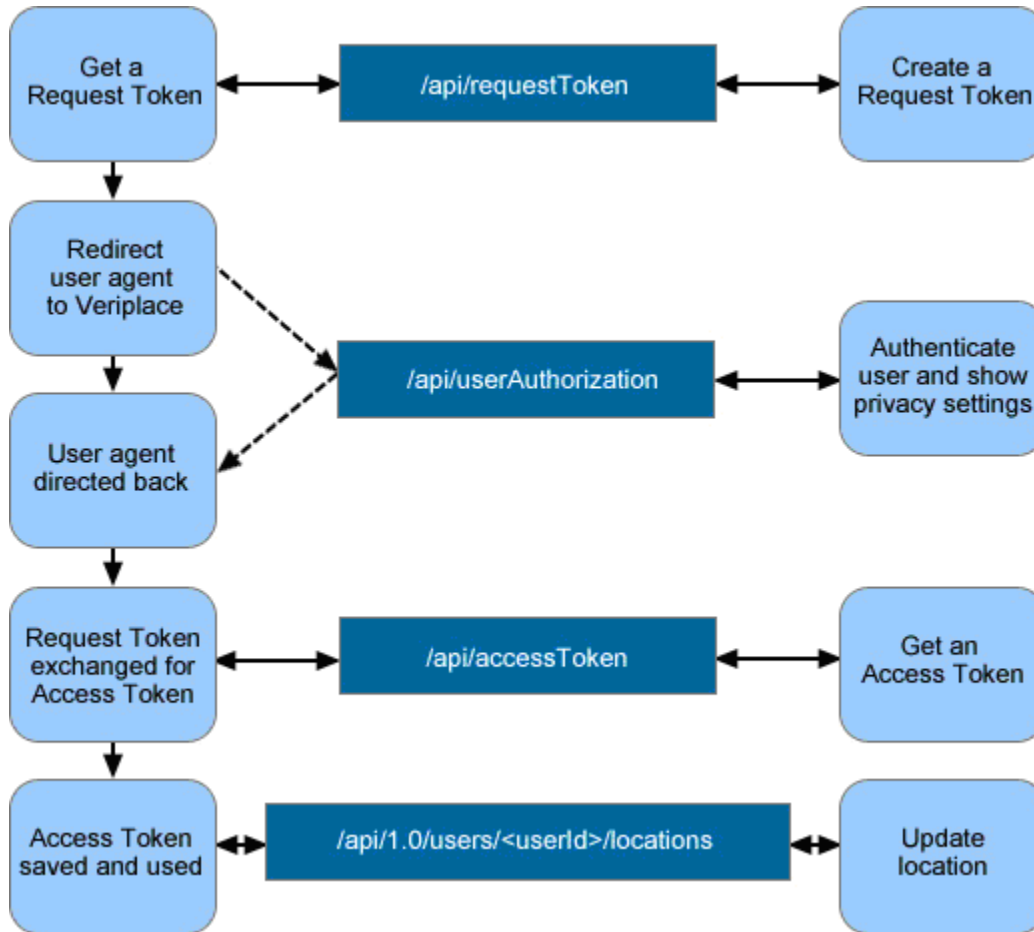
2.3.1.5 OAuth for User Discovery Workflow

The OAuth workflow for user discovery is illustrated below.



2.3.1.6 OAuth for Location Workflow

The OAuth workflow for location access is illustrated below.



2.3.1.7 Getting a Request Token

The developer must obtain a single-use Request Token for the transaction, using the Request Token URL:

```
https://veriplace.com/api/requestToken
```

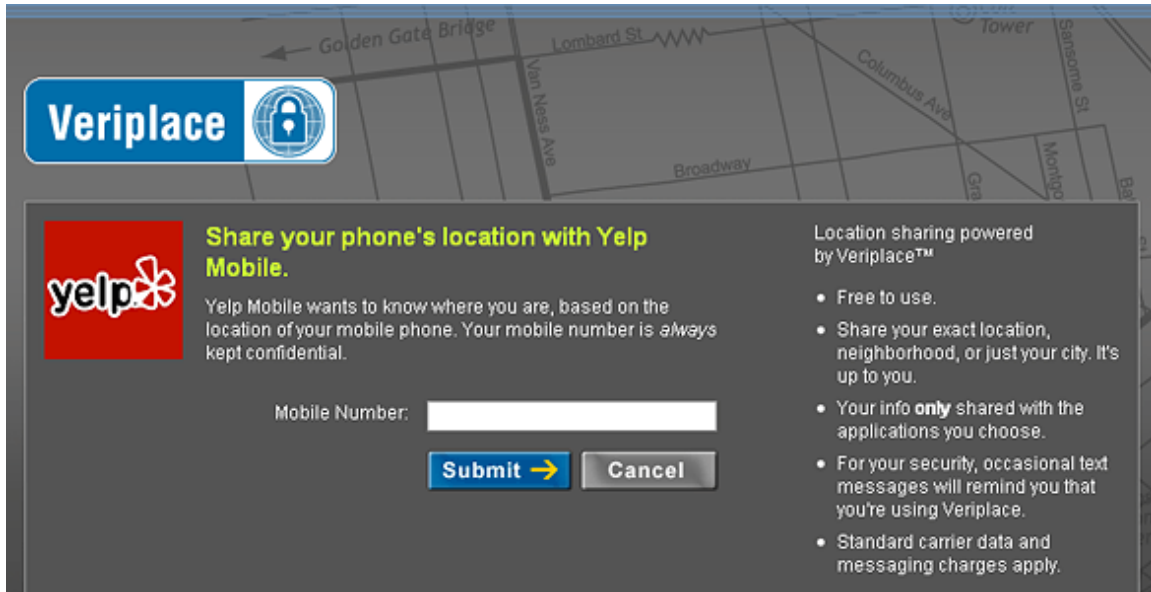
Requests must use HTTP POST and must be signed using the HMAC-SHA1 signature method.

2.3.1.8 Redirecting User Agent for User Authorization

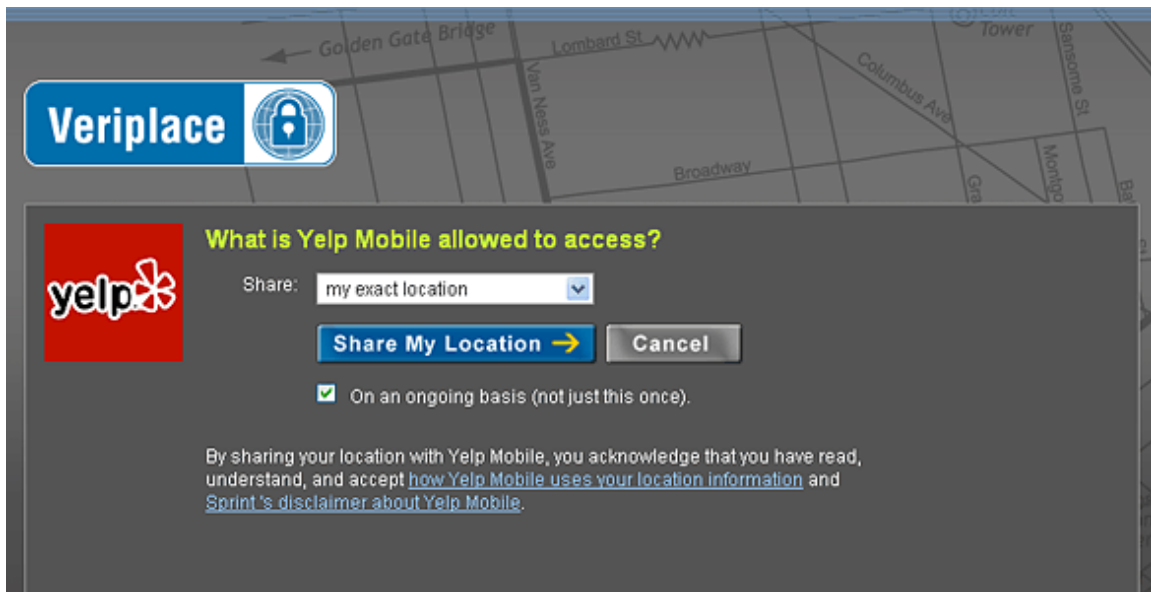
Your application may be required to get authorization from the user before accessing protected resources. Your application gets this authorization by delegating to Veriplace via a redirect to the authorization URL:

```
http://veriplace.com/api/userAuthorization
```

Veriplace prompts the user to approve the request, logging in if necessary, and redirects back to the application. Application branding materials are displayed throughout the interaction, as shown below, to make the process appear seamless.



Users see the screen above during user discovery and the screen below when granting permission to share their location.



When redirecting to Veriplace, the User Authorization URL must encode several parameters, as follows:

- The **oauth_token** parameter identifies the previously obtained Request Token, which will typically be a randomly generated alpha-numeric string, such as **TJTGzFqyx1bTJEourb3V**.
- The **oauth_callback** parameter identifies where the response should be sent upon completion, and could be something like:

```
http://example.com/callback
```

- The Veriplace-specific **uri** parameter identifies the resource for which authorization is requested. This parameter is required.
- The Veriplace-specific **immediate** parameter allows the application to avoid any user interaction. If set to "true", Veriplace will always redirect immediately back to the callback. An Access Token will be immediately available if and only if authorization was previously given. This parameter is optional and defaults to "false".

2.3.1.9 Getting an Access Token

The developer must exchange the Request Token for an Access Token using the following URL:

```
https://veriplace.com/api/accessToken
```

Requests must use HTTP POST and must be signed using the HMAC-SHA1 signature method.

2.3.1.10 Managing Tokens

You may cache Access Tokens, but are not required to do so. Veriplace user authorization requests are idempotent: if authorization has already been granted, the user authorization process will bypass user interaction and immediately redirect back to the callback URL. This property can be used in conjunction with the *immediate* parameter to query for existing authorization.

If you choose to cache Access Tokens, you must ensure that storage is secure. Be aware that some Access Tokens are single-use only, and that all Access Tokens may expire or be revoked.

2.3.2 Protected Resources

User identity and location are OAuth protected resources and are represented using URLs following the REST model. The URL for accessing location is shown below.

```
https://veriplace.com/api/1.1/users/<userId>/locations
```

Note: Terms that appear in brackets are variables, for example, <userId> above.

You are strongly encouraged to use HTTPS for all protected resource URL requests.

2.3.2.1 Protected Resource Encodings: XML and JSON

By default, Veriplace encodes protected resource responses in XML, which is documented using XML Schema. Refer to the [Veriplace XML Schema](#) (page 59).

Applications can also choose to use **JSON**⁵ encoding, by adding the HTTP header "Accept: application/json" to the protected resource request. In most cases, the structure of the JSON data is based on the XML schema, using a one-to-one mapping of XML elements (or attributes) to JSON properties.

The Developer Guide sections describing specific APIs will provide examples of both XML and JSON responses.

2.4 Certifying and Launching Your Application

Once you are confident your application performs the way you want it to, you must submit it to WaveMarket for certification.

There are four steps to certifying and launching your Veriplace-powered application:

1. Prepare your application for final review.
Ensure that you have provided a complete and accurate description of your application on the **Veriplace Developer Portal** as well as up-to-date logos and other branding material. Veriplace uses your application's branding during user authorization to create a better user experience.
2. Notify WaveMarket that your application is ready for testing by clicking the **Publish Application** link within your **Veriplace Developer Portal** account.
Your application must pass our final tests to be certified. You cannot make changes to your application once it is being reviewed.
3. Enter into a commercial agreement with WaveMarket and agree to any additional Terms of Service.
If you have not done so yet, you will be contacted by WaveMarket to discuss final pricing terms and sign a **Pricing Rider**. Depending on the nature of your application, there may be additional Terms to agree to. In some cases, carriers may have to sign off on applications for them to be certified.
4. WaveMarket certifies your application and it becomes available to your public users.

2.5 Billing and Payments

Billing and payments are handled with a pre-paid, declining balance model. Sign in to your developer account on the **Veriplace Developer Portal** to enter credit card information and deposit funds into your account. You can manually refill your account balance at any time. Optionally, you may elect to have your credit card automatically recharged when your balance approaches zero, to avoid interruptions in service.

Each successful **Get Location** request you make for a user's location deducts a small amount from your balance, as described in the **Veriplace Developer Terms of Service** and any **Pricing Riders** agreed upon by you and WaveMarket. If your balance ever runs out, you will not be able to locate end users.

To get you started quickly and let you experiment with the API without having to worry about credit card charges, each developer is provided with a set of simulated test users that can be located **free of charge**.

Notes

1. <http://developer.veriplace.com/devportal>
2. http://www.ctia.org/business_resources/wic/index.cfm/AID/11300

3. <http://oauth.net/>
4. <http://oauth.net/>
5. <http://json.org/>

3. Using the User Discovery API

The Veriplace User Discovery API lets you discover individual user identifiers.

3.1 About User Discovery

Before using any of the other APIs related to users, you must associate users in your application with a Veriplace identity. This process is known as User Discovery.

User identity is considered a protected resource within Veriplace. There are legal and privacy concerns related to exposing personally identifiable information (PII) to third parties, especially in conjunction with location.

To mitigate these concerns, Veriplace employs a randomly generated user identifier in lieu of all other PII data. Veriplace defines its user-oriented APIs in terms of the user identifier, and never exposes other PII data, such as mobile numbers, email addresses or OpenIDs.

3.2 Obtaining a User Identifier

Your application can obtain a user identifier in one of two ways:

- By supplying known PII data, such as an email address, mobile phone number or OpenID
- By requesting that the user log in to Veriplace

Both types of requests require an HTTP GET or POST to the following URL:

```
https://veriplace.com/api/1.2/users
```

Both types of requests must be signed using HMAC-SHA1 and an Access Token, the exact type of which depends on the type of request.

3.2.1 Applications Supply PII Data

If your application users have already supplied their PII data to your application, you can employ the **application-specific** Access Token to sign a protected resource request for the Veriplace user identity. See section [2.3.1.2 OAuth Credentials](#) (page 11) for more

details. The three examples shown below illustrate protected resource requests for email address, phone number and OpenId, respectively:

- User discovery by email address:

```
https://veriplace.com/api/1.2/users?email=user@example.com
```

- User discovery by mobile phone number:

```
https://veriplace.com/api/1.2/users?mobile=1115551212
```

Note: A North American mobile number must contain 10 decimal digits, optionally prefixed by "1" or "+1"; all other non-digit characters are ignored, so "4155551212", "+14155551212", and "(415) 555-1212" are all equivalent.

- User discovery by OpenID:

```
https://veriplace.com/api/1.2/users?openid=user@openid.example.com
```

Note that the first three steps of the four-step OAuth process are not needed here because the developer obtains the **application-specific** Access Token during registration.

It is also possible to query for multiple users by PII using multiple email address, mobile phone numbers, or OpenID parameters:

```
https://veriplace.com/api/1.2/users?mobile=1115551212&mobile=1115551213
```

To avoid URL length limits, HTTP POST should be used if multiple PII parameters are specified.

3.2.2 Applications Request User Log in to Veriplace

The other option for User Discovery is to redirect the user to Veriplace and have them log in to identify themselves. In this case, Veriplace returns a one-time Access Token, which is used to sign the protected resource request.

For User Discovery, the User Authorization URL must specify the `uri` parameter as defined below:

```
https://veriplace.com/api/userAuthorization?uri=https://veriplace.com/api/1.2/users
```

This approach will not always require a user log in. In some cases the user may already be logged in or may have set a cookie that allows them to avoid login for user discovery.

3.2.3 Common Failure Conditions

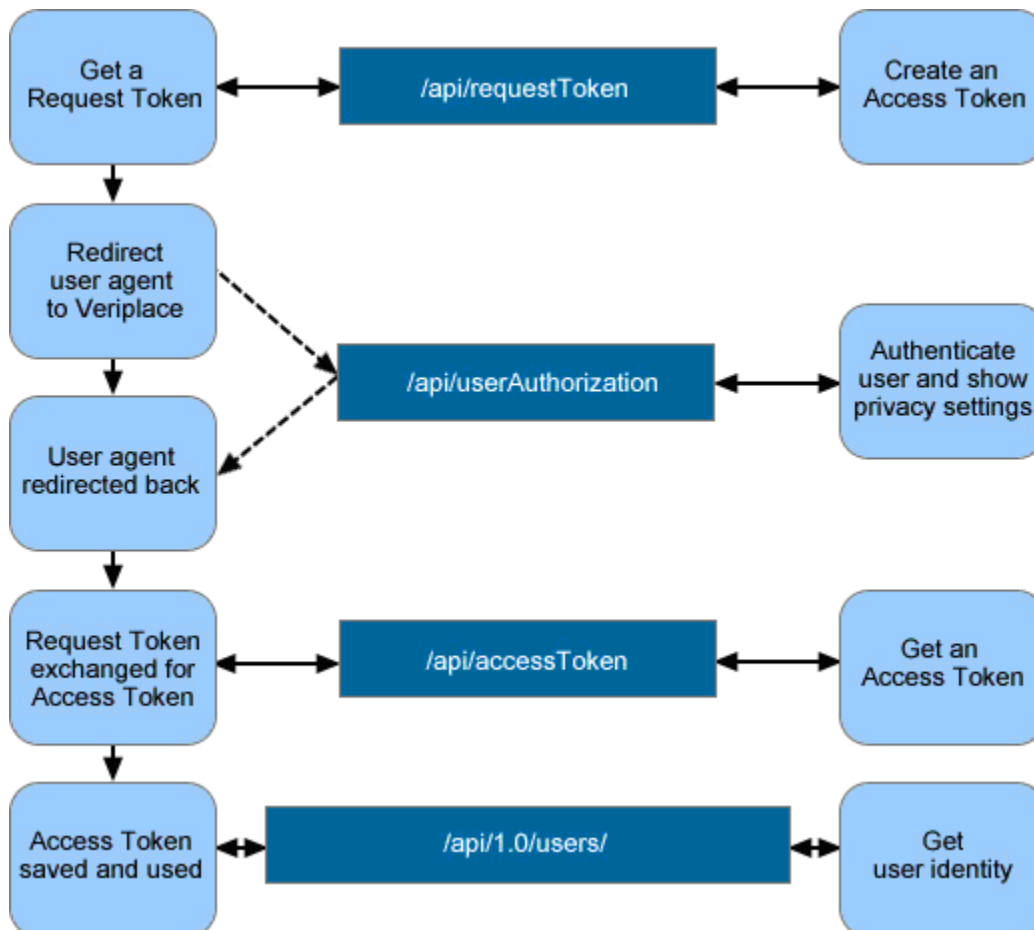
User discovery will fail if the user chooses not to log in or if the provided PII are not recognized.

3.2.4 Registering New Users

The User Discovery API handles users who are not currently Veriplace subscribers; users may register during the user authorization redirect; registration will be invisible to your application.

3.2.5 User Discovery Sequence Diagram

There are five steps of communication that occurs between your application and Veriplace during the process of discovering a single user via login. These are illustrated in the sequence diagram below.



The sequence of events illustrated above is described below.

1. Your application requests an OAuth Request Token using the Veriplace Request Token URL, which is granted.
2. Veriplace redirects your application to the Veriplace user authorization URL for secure authentication.
3. Veriplace redirects the User Agent back to your application after the user logs in.
4. Your application exchanges the Request Token for an OAuth Access Token, which is granted.

5. Your application uses the Access Token to obtain the identity of the user who logged in.

At the end of this process, your Access Token is revoked.

3.3 Successful Requests

On success, you will receive an HTTP 200 ("OK") code and XML content representing the user identifier or user identifiers queried.

If the Application requests that the user login or only one user is queried using PII data, a single result will be returned. The XML encoding of the result is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<user id="1378603015410979905" xmlns="http://veriplace.com/xml/1.2" />
```

If you requested JSON encoding, the same result would be encoded like this:

```
{ "user": { "id": 1378603015410979905 } }
```

Alternatively, if multiple PII parameters are specified, a result will be returned for every parameter that was matched to a user identifier. The XML encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
<identifiedUsers xmlns="http://veriplace.com/xml/1.2">
  <identifiedUser key="1115551212" keyType="mobile"><user
id="1378603015410979905"/></identifiedUser>
  <identifiedUser key="1115551213" keyType="mobile"><user
id="7799817486972445687"/></identifiedUser>
</users>
```

The JSON encoding for a multiple-user result:

```
{ "identifiedUsers": [
  { "key": "1115551212", "keyType": "mobile",
    "user": { "id": 1378603015410979905 } },
  { "key": "1115551213", "keyType": "mobile",
    "user": { "id": 7799817486972445687 } }
] }
```

3.4 Unsuccessful Requests

In accordance with the **HTTP Response Codes** section of the OAuth specification, you will receive an HTTP 400 ("Bad Request") or 401 ("Unauthorized") code if something went wrong with the OAuth signature. In some cases, a 401 error code will be accompanied by the reason phrase "Invalid / non-monotonic timestamp", which is commonly a sign that your system clock is misconfigured.

In accordance with REST, an HTTP 404 (Not Found) error message is returned if no user could be identified.

4. Using the Get Location API

You can use the Get Location API to obtain location information for a particular user.

Before using the Get Location API, you must obtain a Veriplace user identifier so you can request permission to locate the user.

4.1 About Location

Veriplace models location as both a geographical position and an accuracy measurement. The degree of accuracy available depends on three things – 1) the degree the user wants to reveal, 2) the degree that can be obtained by the carrier, and 3) the **location mode** being used by the application - see [4.2.6 Using Location Modes](#) (page 27).

When thinking about location, keep in mind the following considerations:

- User preference is primary. If users limit access or accuracy, your application cannot force access or obtain a greater degree of accuracy.
- If users turn their phones off, current location information will not be available. In these cases, cached location information, or location information recorded during an earlier connection, may be available if the user permits its use.
- Carriers vary in their ability to pinpoint their users. Location derived from a single cell tower is less accurate than location derived from cell triangulation (two or more cell towers) or from GPS.
- Obtaining location information involves a tradeoff of time, cost and accuracy. For example, the most accurate location information - that based on GPS - usually takes the most time to obtain and often incurs the highest cost. The least accurate information - cell tower – usually takes the least time to obtain and as often available more cheaply.

4.2 Obtaining User Location

Your application can obtain location information for a mobile user whose identity you have previously obtained. You may make the request in one of three ways:

- a synchronous request which blocks until the result is ready;
- an asynchronous request which returns immediately, specifying a callback URL for Veriplace to push the result to your application;

- an asynchronous request which returns immediately, followed by a polling query to check for the result later.

All of these methods require an HTTP POST to a protected resource URL. Requests must be signed using HMAC-SHA1 and an Access Token representing the user's permission to be located. You can obtain the Access Token interactively, using the OAuth User Authorization process (in which case the `uri` parameter for authorization should be the same as the synchronous location request URL); or use the [Verify Permission API](#) (page 40) to look up an existing Access Token for a user.

For synchronous requests, use the following URL:

```
https://veriplace.com/api/1.2/users/<userId>/locations
```

For asynchronous requests, use the following URL:

```
https://veriplace.com/api/1.2/request/users/<userId>/locations
```

The following optional parameters may be added to the URL query string:

- `mode`: the name of the location mode to use. See [4.2.6 Location Modes](#) (page 27).
- `callback`: for asynchronous requests only, a URL where your application can receive HTTP an notification from Veriplace. See [4.5 Getting Asynchronous Results](#) (page 35). Note that this must be a valid externally accessible URI; it cannot be at `localhost`.
- `callbackContentType`, `callbackFormParamName`: for asynchronous requests only, these parameters control the format of the callback notification. See [4.5 Getting Asynchronous Results](#)¹.
- `noCachedPosition=true`: add this parameter if you do *not* want Veriplace to include a recent position in the error response if a new position is unavailable. See [4.4 Unsuccessful Location Requests](#) (page 34).
- `withNearestPOI=true`: add this parameter if you want Veriplace to search for a nearby "point of interest". See [4.2.9 Points of Interest](#) (page 29).
- `withNearestIntersection=true`: add this parameter if you want Veriplace to search for a nearby street intersection. This feature is not enabled for all applications. See [4.2.9 Points of Interest](#) (page 29).

4.2.1 Accuracy

The accuracy for location information is determined by two measurements:

- Coordinates for longitude and latitude.
- An integer signifying the radius of uncertainty in meters.

The first measurement specifies the proximate location of the user, while the second measurement indicates an area around that location where the user is believed to be with a known probability.

Applications cannot require or specify a degree of accuracy for the following reasons:

- Carriers can employ a range of methods to locate users that vary in their accuracy.
- Users can opt to turn off location-finding for their device.
- Users can modify location-finding accuracy and limit the results to a general area, such as a city.

4.2.2 Relevance

Relevance refers to the timeliness of location information. Location information can be obtained in two formats – position determination (real-time) and cached. Position determination is more relevant, but this method may take longer to obtain location information.

Cached location information can be retrieved very quickly, and some location request strategies may optimize the retrieval so the information is recent and relevant.

4.2.3 Latency

Latency refers to the delay in obtaining location information after requesting it. Obtaining user location information can take time. Some position determination technologies, such as cell tower position determination, usually take very little time. Other methods need more time. For example, an A-GPS request typically takes 30 seconds but can take up to 90 seconds in extreme conditions, in part because A-GPS must retrieve almanac and ephemeris data from the network, obtain sufficiently strong satellite signals, perform the positioning computation, and finally return the results back to the requestor.

Applications that require low latency requests are encouraged to pick their location mode accordingly.

4.2.4 User Location Restrictions

Developers should be aware that users may want to restrict when and how they can be located. Some options Veriplace gives users include:

- **Do not locate me more than # times per day/week/month.**
- **Do not locate me too frequently.**

When the limits set by the user are met, location information cannot be retrieved for that user until the specified time period has elapsed. In these cases, the response message indicates the user has prevented the release of their location. Your application can use this code to prevent reiterative requests for more information until a certain amount of time has elapsed.

- **Do not/only locate me at certain times of day.**

The user's device will not respond if time and dates windows for access have been specified by the user. Your application will get an error code indicating that access to location was restricted for privacy reasons.

- **Do not locate me with too much accuracy.**

The user can ask that their location be provided with less accuracy.

- **Locate me only once before asking permission again.**

The user can ask that you apply for permission each time you try to locate the user. In this case, the returned Access Token will not be valid for additional location requests.

4.2.5 Common Failure Conditions

The user cannot be located if any of the following conditions apply:

- User's device is turned off or outside of coverage.
- User has restricted access to their location.
- Position determination fails to locate the user.

In these cases, an appropriate error code is returned so that the application can determine whether to try again or wait a suitable amount of time before re-trying.

4.2.6 Location Modes

Veriplace uses a variety of methods to identify the locations of users, including single cell tower, cell tower triangulation, and GPS among others. It is possible that wireless carriers will provide some methods but not others. Each method has advantages and drawbacks. For example, some are more accurate but slower while others are faster but cost more. Some methods are not always available at the same level of quality. For example, users in a rural area will get much lower accuracy from cell tower location because cell towers tend to be spread further apart in rural areas, while GPS can be imprecise in the densest urban environments due to reflection and interference from buildings.

To reduce the complexity of choices faced by your application, Veriplace provides the capacity to select the most appropriate mode of obtaining location information for your application. Veriplace currently provides a default mode that is **free** to use:

freedom	Freedom mode is Veriplace's default location mode. This basic mode lets developers locate smartphone users for free. Most GPS enabled smartphones are supported, though there are no guarantees regarding the relevance or accuracy of the data.
----------------	--

In addition, Veriplace currently provides two modes for professional access to location. These are intended for developers who are willing to pay for greater coverage rather than be limited to only locating smartphones:

zoom	Zoom mode is suited for applications that want to determine, as precisely as possible, where a user is now, regardless of phone type. Zoom mode returns the highest accuracy possible for a given user and situation. Accuracy is optimized at the expense of higher latency and cost.
area	Area mode is suited for applications that only need to know the general area a user is presently in, typically at neighborhood- or city-level accuracy. Like zoom mode, area mode is not limited to smartphones. Latency and cost are reduced at the expense of accuracy.

You can specify the mode you want to use with the following URL, which uses the zoom location mode as an example:

```
https://veriplace.com/api/1.0/users/<userId>/locations?mode=zoom
```

4.2.7 Expiration Dates

The expiration date for permission to use a user's location data is specified in the location response XML encoding using the data element named *expires* as defined below.

```
<xs:element name="expires" type="xs:dateTime" />
```

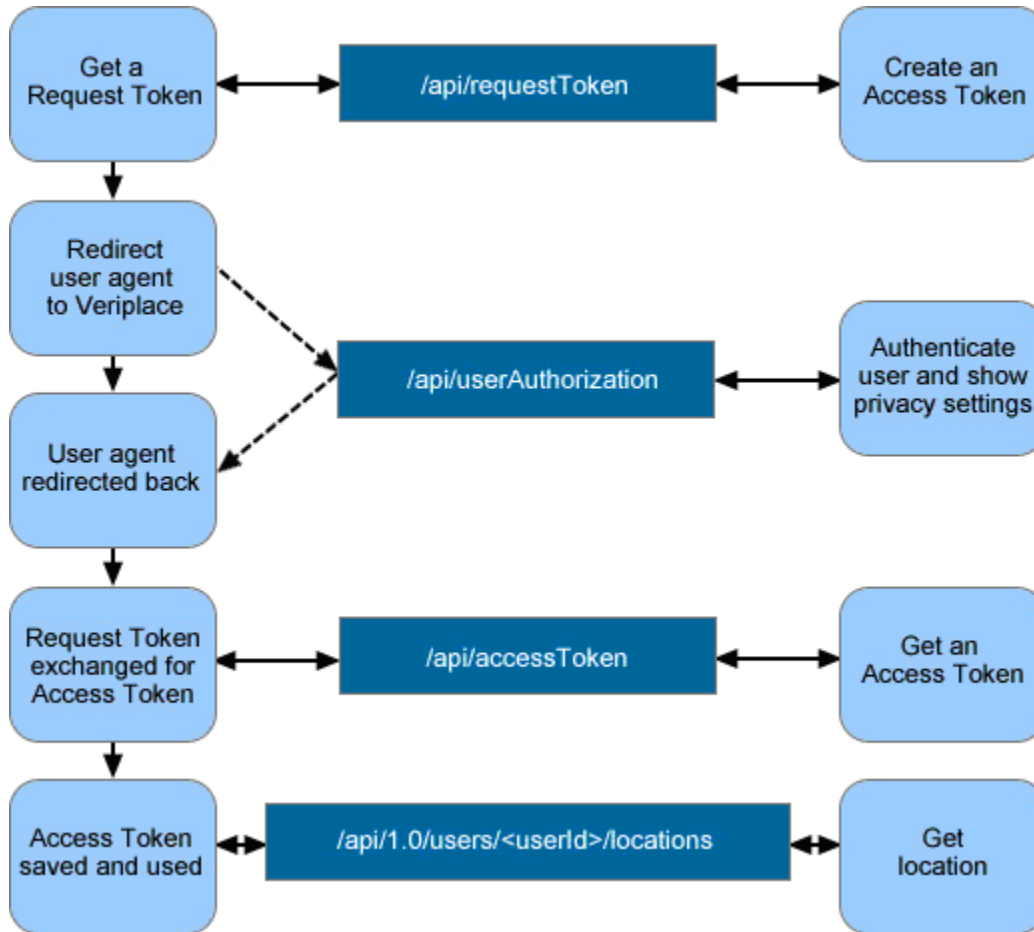
Every XML document containing location data that Veriplace returns indicates the expiration time of these data with an *expires* element, as shown below.

```
<expires>2008-01-14T23:06:39Z</expires>
```

Your application must not use or retain location data past its expiration date.

4.2.8 Get Location Sequence Diagram

There are five steps of communication that occurs between your application and Veriplace during the process of trying to obtain location information on a single user. These are illustrated in the sequence diagram below.



The sequence of events illustrated above is described below.

1. Your application requests an OAuth Request Token using the Veriplace Request Token URL, which is granted.
2. Your application redirects the User Agent to the Veriplace user authorization URL.
3. Your application is re-directed back with permission.
4. Your application exchanges the Request Token for an OAuth Access Token, which is granted.
5. Your application uses the Access Token to obtain location information for a single user.

When the process is complete, your Access Token may or may not be reused, depending on the level of permission granted in Steps 2 and 3.

4.2.9 Points of Interest and Street Intersections

Using the optional `withNearestPOI=true` parameter, you can ask Veriplace to query a map service for the nearest "point of interest" to the user's location. A point of interest can be a business or other landmark that could serve as a point of reference for describing the user's whereabouts. Veriplace will provide the following information:

- name
- category (e.g. "Restaurants")
- position (using the same format as the user's location)
- phone number, if available
- distance from the user's location, in meters
- heading from the user's location, in degrees clockwise from due north

Similarly, you can also specify the parameter `withNearestIntersection=true`, to query the nearest street intersection to the user's location. If available, this is returned as a position in the same format as a user location, but with two street names. Note that this feature (unlike the point of interest query) is currently *not* enabled by default, because the underlying service has higher overhead; please contact WaveMarket if you would like to use it.

4.3 Successful Requests

If a *synchronous* location request is successful, you will receive an HTTP 200 ("OK") code and a response representing the user's location. This has the following properties:

- A unique ID specific to this location request.
- The date and time that the location was generated.
- An expiration date after which the location must be discarded.
- Longitude and latitude, measured in degrees.
- An uncertainty radius, measured in meters.
- Optional geographic address fields: street address, city, neighborhood, state, postal code, and country code. Veriplace attempts to calculate these for every location, but they will not always be available.

The XML encoding of a location response, when all of the above properties are present, looks like this. Note that `<address>`, or any element within `<address>`, may be absent.

```
<?xml version="1.0" encoding="UTF-8"?>
  <location xmlns="http://veriplace.com/xml/1.2" id="1234">
    <created>1970-01-14T23:06:39Z</created>
    <expires>1970-01-14T23:06:39Z</expires>
    <position>
      <longitude>-122.123456</longitude>
      <latitude>37.123456</latitude>
      <uncertainty>100.0</uncertainty>
      <address>
        <street>5858 Horton St</street>
        <city>Emeryville</city>
        <neighborhood>Downtown</neighborhood>
        <state>CA</state>
        <postal>94608</postal>
      </address>
    </position>
  </location>
```

```

        <countryCode>USA</countryCode>
    </address>
</position>
</location>

```

The same response in JSON encoding:

```

{
  "location": {
    "id": 1234,
    "created": "1970-01-14T23:06:39Z",
    "expires": "1970-01-14T23:06:39Z",
    "position": {
      "longitude": -122.123456,
      "latitude": 37.123456,
      "uncertainty": 100.0,
      "address": {
        "street": "5858 Horton St",
        "city": "Emeryville",
        "neighborhood": "Downtown",
        "state": "CA",
        "postal": "94608",
        "countryCode": "USA"
      }
    }
  }
}

```

If Veriplace includes a nearby [point of interest](#) (page 29) in response to the `withNearestPOI` parameter, the following elements are added:

```

<?xml version="1.0" encoding="UTF-8"?>
  <location xmlns="http://veriplace.com/xml/1.2" id="1234">

    <!-- created, expires, position: same as above -->

    <nearPointOfInterest>
      <pointOfInterest>
        <name>Amtrak Station</name>
        <category>Train Stations</category>
        <position>
          <longitude>-122.12344</longitude>
          <latitude>37.123456</latitude>
          <!-- may also include address -->
        </position>
        <phone>111-555-1212</phone>
      </pointOfInterest>
      <distance>100.0</distance>
    </nearPointOfInterest>
  </location>

```

```

        <heading>270.0</heading>
    </nearPointOfInterest>

</location>

```

```

{
  "location": {

    /* (id, created, expires, position: same as above) */

    "nearPointOfInterest": {
      "pointOfInterest": {
        "name": "Amtrak Station",
        "category": "Train Stations",
        "position": {
          "longitude": -122.12344,
          "latitude": 37.123456
        },
        "phone": "111-555-1212"
      },
      "distance": 100.0,
      "heading": 270.0
    }
  }
}

```

If Veriplace includes a street intersection in response to the `withNearestIntersection` parameter, the following elements are added:

```

<?xml version="1.0" encoding="UTF-8"?>
  <location xmlns="http://veriplace.com/xml/1.2" id="1234">

    <!-- created, expires, position: same as above -->

    <nearIntersection>
      <intersection>
        <position>
          <longitude>-122.12344</longitude>
          <latitude>37.123456</latitude>
          <address>
            <street>Hollis St.</street>
            <street2>59th St.</street2>
            <!-- also includes city, state, etc. -->
          </address>
        </position>
      </intersection>
      <distance>100.0</distance>
      <heading>270.0</heading>
    </nearIntersection>
  </location>

```

```

    </nearIntersection>
  </location>

```

```

{
  "location": {
    /* (id, created, expires, position: same as above) */

    "nearIntersection": {
      "intersection": {
        "position": {
          "longitude": -122.12344,
          "latitude": 37.123456,
          "address": {
            "street": "Hollis St.",
            "street2": "59th St."
          }
        }
      }
    },
    "distance": 100.0,
    "heading": 270.0
  }
}

```

4.3.1 Successful Asynchronous Requests

For an *asynchronous* location request, if the user ID and Access Token were valid, you will receive an HTTP 200 code and a unique identifier, or *nonce*, representing the location request. You can use this value as an association key later when you get the result; see [4.5 Getting Asynchronous Results](#) (page 35). The XML encoding of this value is as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
  <getLocationResult xmlns="http://veriplace.com/xml/1.2">
    <nonce value="request ID"/>
    <user id="user ID"/>
  </getLocationResult>

```

Or, in JSON encoding:

```

{ "getLocationResult": {
  "nonce": { value: "request ID" },
  "user": { id: "user ID" }
} }

```

4.4 Unsuccessful Location Requests

In accordance with the *HTTP Response Codes* section of the OAuth specification, you will receive an HTTP 400 ("Bad Request") or 401 ("Unauthorized") code if something went wrong with the OAuth signature. In some cases, a 401 error code will be accompanied by the reason phrase "Invalid / non-monotonic timestamp", which is commonly a sign that your system clock is misconfigured.

In addition, successful HTTP responses may still return several possible errors (defined by Veriplace XML Schema) that can occur when trying to obtain location. The errors are marked by a **position error** element in the response, which replaces the **position** element.

The position error element may (but will not always) contain a **cached position** element representing the last known successful location result for this user. If you do *not* want this information, you can suppress it by passing the parameter `noCachedPosition=true`. In XML, a response with an error and a cached position looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
  <location xmlns="http://veriplace.com/xml/1.2" id="1234">
    <created>1970-01-14T23:06:39Z</created>
    <expires>1970-01-14T23:06:39Z</expires>
    <positionError>
      <code>100</code>
      <message>Subscriber could not be located.</message>
      <cachedPosition>
        <longitude>-122.123456</longitude>
        <latitude>37.123456</latitude>
        <uncertainty>100.0</uncertainty>
      </cachedPosition>
    </positionError>
  </location>
```

The same response in JSON encoding:

```
{
  "location": {
    "id": 1234,
    "created": "1970-01-14T23:06:39Z",
    "expires": "1970-01-14T23:06:39Z",
    "positionError": {
      "code": 100,
      "message": "Subscriber could not be located.",
      "cachedPosition": {
        "longitude": -122.123456,
        "latitude": 37.123456,
        "uncertainty": 100.0
      }
    }
  }
}
```

4.4.1 Position Failure (100)

This error code will be returned if location could not be obtained because, for example, the mobile device is turned off, out of service or unable to get a good GPS signal indoors.

4.4.2 Restricted (200)

This error code will be returned if the user has restricted access to their location. Access should be available at another time.

4.4.3 Permission Denied Message

In the event of an HTTP 401 ("Unauthorized") code, the provided Access Token should be considered invalid (or may have been revoked by the user). In this case, the Access Token should be discarded and user authorization obtained again.

4.4.4 Billing Not Authorized Message

In the event that a Veriplace location request incurs a cost and you do not have sufficient credit to cover this cost, you will receive an HTTP 403 ("Forbidden") code. In this case, additional credit must be added to your account balance.

4.5 Getting Asynchronous Results

As described above ([4.2 Obtaining User Location](#) (page 24)), you may choose to post a location request to Veriplace and proceed with other tasks, rather than waiting until the location is available. This decision depends only on what is convenient for the architecture of your application; Veriplace processes the location request the same way in either case, the only difference is in how it delivers the result.

4.5.1 Querying Results

If you make an asynchronous request and do not specify a `callback` parameter, then it is up to you to query Veriplace for the result at some point in the future. You can do this by making an HTTP GET request to the following protected resource URL, using the same `nonce` value that was returned when you made the original request:

```
https://veriplace.com/api/1.2/requests/<request ID>
```

You can repeat the query as many times as necessary, for up to 24 hours; after that, the request expires and any further queries with that `nonce` value will receive an HTTP 404 error.

If the request has not yet completed, Veriplace returns an HTTP 204 status ("No Content"). If the request has completed, Veriplace returns an HTTP 200 status and an XML or JSON response which describes both the original request and the result, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
  <getLocationResult xmlns="http://veriplace.com/xml/1.2">
    <nonce value="request ID"/>
    <user id="user ID"/>
    <mobileNumber>mobile number</mobileNumber>
    <location id="location ID">
      <!-- location fields: see above -->
    </location>
  </getLocationResult>
```

```
{
  "getLocationResult": {
    "nonce": { "value": "request ID" },
    "user": { "id": "user ID" },
    "mobileNumber": "mobile number",
    "location": { /* location fields: see above */ }
  }
}
```

If the location request failed because the device was not locatable, then the message format will still be the same, except that the `<location>` element will contain a `<positionError>` rather than a `<position>`. See [4.4 Unsuccessful Location Requests](#) (page 34). If the request failed for a different reason -- one that would have returned an HTTP error code if you had made the request synchronously -- then instead of a `<location>`, there will be an error element in this format:

```
<apiError>
  <code>403</code>
  <message>Billing Not Authorized</message>
</apiError>
```

```
"apiError": {
  "code": 403,
  "message": "Billing Not Authorized"
}
```

4.5.2 Receiving Results by Callback (Push)

If your application can receive HTTP requests, you can tell Veriplace to push results to you at a URL of your choice, which you specify as the `callback` parameter of the asynchronous location request. Veriplace will make an HTTP POST request to this URL. This message can use one of several formats, depending on what is most convenient for your application:

1. The *summary* format, which is used if you only specified a callback URL, contains a query string in the `application/x-www-form-urlencoded` content type. The summary message does not contain the actual location data; instead, use the value of the

`location` parameter to retrieve the location data using the [Past Location API](#) (page 38). The query string parameters are as follows:

- `nonce`: the unique identifier that was returned when you made the original request
 - `user`: the Veriplace user ID
 - `code`: the HTTP status code representing the outcome of the request (200 if successful)
 - `reason`: an optional error description, equivalent to the HTTP status message
 - `location`: the Veriplace location ID, if the request was successful
2. The *extended* format is used if you also specified a `callbackContentType` parameter, whose value must be either `"text/xml"` or `"application/json"`. In this case, the message will be a `<getLocationResult>` object in XML or JSON, containing all the location details if available, as described in [4.5.1 Querying Results](#) (page 35). To use this mode, your callback URL must be secure (`https:`); Veriplace will not send protected information such as location details to an insecure URL.
 3. The *form-encoded extended* format is the same as the extended format, but wraps the entire XML or JSON document in a form-encoded parameter -- e.g., `<?xml version=... becomes myParamName=%3C%3Fxml+version%3D...`. This format is used if you specify `callbackFormParamName=myParamName` along with `callbackContentType`. This may be useful if your application receives callbacks through a web gateway that can only accept form-encoded content, but you still want the details included in the extended format.

To confirm receipt of the callback, your application must return an HTTP 200 response within 2000 milliseconds. Otherwise, Veriplace will consider the request a failure and will retry up to 3 times before giving up.

Notes

1. http://localhost:8888/devportal/developerguide/location-api/edit/301f2f032c1d3b7f69225f790b087f6f0a251609/part-SimpleDocumentContent#devguide_location_api_async_results

5. Using the Past Location API

You can use the Past Location API to obtain location that you obtained previously for a specific Veriplace user.

Before using the Past Location API, you must possess a Veriplace user identifier, a Veriplace location identifier, and an Access Token permitting you to obtain the user's location.

5.1 About Past Location

Every location returned by Veriplace includes a unique identifier. Developers may access previously obtained location information in real time using this identifier.

The Past Location API exists primarily to relieve applications of the responsibility of storing location data.

5.2 Retrieving Past Location

Veriplace represents past location information as an HTTP GET to the following resource URL:

```
https://veriplace.com/api/1.2/users/<userID>/locations/<locationID>
```

Requests must be signed using HMAC-SHA1 and an Access Token permitting you to locate the user.

You may also add either or both of the following query string parameters to query related location information, as described in [4.2.9 Points of Interest and Street Intersections](#) (page 29):

- `withNearestPOI=true`
- `withNearestIntersection=true`

You can query this information even if the original location request did not include those parameters, but note that the query may not succeed (in which case the corresponding elements will not appear in the location response), and that the street intersection parameter is not enabled for all applications.

5.2.1 Common Failure Conditions

Past location data will not be available after it expires or if the user manually deletes it.

5.3 Successful Requests

On success, you will receive an HTTP 200 ("OK") code, and a location response in the same format used by the [Get Location API](#) (page 24).

5.4 Unsuccessful Requests

In accordance with the *HTTP Response Codes* section of the OAuth specification, you will receive an HTTP 400 ("Bad Request") or 401 ("Unauthorized") code if something went wrong with the OAuth signature. In some cases, a 401 error code will be accompanied by the reason phrase "Invalid / non-monotonic timestamp", which is commonly a sign that your system clock is misconfigured.

6. Using the Verify Permission API

You can use the Verify Permission API to query an existing Access Token for the Get Location API: either retrieving the Access Token associated with your application for a particular mobile user, or checking the validity of an Access Token that you had stored locally.

The Verify Permission API only works if your application already has permission to locate the given user; it does not grant new permissions.

6.1 Querying Permissions by User

Your application can request the Access Token and token secret for locating a particular mobile user, given the user ID. This is an efficient alternative to the OAuth redirect process for user authorization, as long as permission has already been granted.

To make this request, submit an HTTP GET to the following URL:

```
https://veriplace.com/api/1.2/permission/users/<userID>/locations
```

The user ID is the unique Veriplace identifier for the mobile user, the same one that you can obtain from the User Discovery API and pass to the Get Location API. (Note that this URL is identical to the URL for locating a user in the Get Location API, except for the `/permission/` prefix.)

The request must be signed using HMAC-SHA1 and the application-specific Access Token (the one configured for your application as a whole, rather than for a particular user).

6.1.1 Successful Requests

On success, you will receive an HTTP 200 ("OK") code, and a response document describing the Access Token and token secret. The default XML encoding is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<token value="tokenValue" secret="tokenSecret"
xmlns="http://veriplace.com/xml/1.2" />
```

You can also request a JSON-encoded response by adding the header `Accept: application/json` to the request. The JSON encoding for the response is as follows:

```
{ "token": { "value": "tokenValue", "secret": "tokenSecret" } }
```

6.1.2 Unsuccessful Requests

You will receive a HTTP 401 status code if your application does not have permission to locate the user, or 404 if the user ID is unknown.

6.2 Verifying Permissions by Token

If your application has previously obtained and stored an Access Token for locating a user, you may choose to verify that the permission is still valid before using it, since location permission could have been revoked by the end user.

To verify a permission when you already have the Access Token, submit an HTTP GET to the following URL:

```
https://veriplace.com/api/1.2/permission
```

Requests must be signed using HMAC-SHA1 and the Access Token representing the permission.

6.2.1 Successful Requests

On success, you will receive an HTTP 200 ("OK") code.

6.2.2 Unsuccessful Requests

You will receive a HTTP 401 status code if the permission was already deleted or is otherwise invalid.

7. Using the Permission Delete API

You can use the Permission Delete API to revoke your ability to obtain location information for an individual mobile user. If you don't delete a permission, it will expire only if the user chooses to revoke it.

Before using the Permission Delete API, you must possess a Veriplace user identifier and an Access Token representing the permission that you want to delete.

7.1 About Permissions

Permission is required to obtain the location information of a mobile user. Your application obtains permission using the OAuth process. An individual permission is represented by an Access Token.

Permissions may be deleted in a variety of ways. Some permissions expire immediately after their use, while others can last indefinitely or at least until the user changes their privacy settings.

You can choose to delete a permission. For example, your application may support cancellation of service and you may wish to surrender your rights to obtain location for a user when this happens. This chapter describes how to delete such a permission.

7.2 Deleting Permissions

To delete a permission, submit an HTTP POST or DELETE to the following URL:

```
https://veriplace.com/api/1.2/users/<userId>/permissions
```

Requests must be signed using HMAC-SHA1 and the Access Token representing the permission.

7.2.1 Common Failure Conditions

You cannot delete permissions that are not associated with your application or that were already deleted.

7.3 Successful Requests

On success, you will receive an HTTP 200 ("OK") code.

7.4 Unsuccessful Requests

In accordance with the *HTTP Response Codes* section of the OAuth specification, you will receive an HTTP 400 ("Bad Request") or 401 ("Unauthorized") code if something went wrong with the OAuth signature. In some cases, a 401 error code will be accompanied by the reason phrase "Invalid / non-monotonic timestamp", which is commonly a sign that your system clock is misconfigured.

You will also receive a 401 error code if the permission cannot be deleted or was already deleted.

8. Using the Get Permissions API

You can use the Get Permissions API to determine the list of Veriplace user identifiers that are currently granting permission to your application to access location.

Before using the Get Permissions API, you must possess the application-specific Access Token. See section [2.3.1.2 OAuth Credentials](#) (page 11) for more details.

8.1 About Getting Permissions

Veriplace retains a record of every permission granted by a user to be located by each application, including permissions granted by users without your application's direct knowledge.

Your application can choose to periodically query the current list of permissions to synchronize its state with that of the Veriplace server.

8.2 Getting Permission

To get permissions, submit an HTTP GET to the following URL:

```
https://veriplace.com/api/1.2/permissions
```

Requests must be signed using HMAC-SHA1 and the application-specific Access Token.

Requests may optionally specify a "max" parameter to limit the number of results returned. Requests that specify the "max" parameter may also specify a "first" parameter to page through results.

8.3 Successful Requests

On success, you will receive an HTTP 200 ("OK") code and a list of user identifiers. The XML encoding looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<users xmlns="http://veriplace.com/xml/1.2">
  <user id="1378603015410979905"/>
  <user id="7799817486972445687"/>
</users>
```

The same list in JSON encoding:

```
{
  "users": [
    { "id": "1378603015410979905" },
    { "id": "7799817486972445687" }
  ]
}
```

8.4 Unsuccessful Requests

You will receive a HTTP 400 status code if the request was malformed.

9. Using the Invitation API

Not every application fits naturally into an OAuth redirect-based authorization process. For example, SMS-based applications have no natural way to send a redirect. For these situations, Veriplace offers an SMS-based opt-in, in which the end user replies affirmatively to an SMS invitation to grant permission to be located.

The Invitation API allows authorized applications to invite a specific mobile number to opt-in to be located. The calling application may choose to receive an asynchronous callback when this opt-in occurs, or may poll for the result.

This feature must currently be enabled by WaveMarket for specific applications based on a developer request. Please contact WaveMarket developer support if you plan to use the Invitation API. Applications under development that have not yet been published will only be able to invite users who are on their Test Phones list (see [Enabling Developer Mode!](#)).

9.1 About SMS Opt-In

The SMS Opt-In mechanism consists of Veriplace sending an SMS to the end user asking them to grant permission to be located by a specific application. The user can send an SMS reply to grant permission, and also to register with Veriplace at the same time if not already registered; or the user can refuse, either by declining that particular invitation or by requesting a block on all future SMS invitations.

There is also a user-initiated SMS opt-in mechanism, separate from the Invitation API: your application can be configured with a unique keyword which the user can send to Veriplace's SMS shortcode, in which case the user can grant permission without requiring any action by your application. Contact WaveMarket if you would like to enable this feature.

9.2 Sending an Invitation

Invitations may be requested by submitting an HTTP POST to:

```
https://veriplace.com/api/1.2/invitations
```

Requests must be signed using HMAC_SHA1 and the **application-specific** Access Token.

The request's form-encoded body contains the following parameters:

- `mobile` (required): the mobile number to be invited. Veriplace accepts ten-digit numbers optionally prefixed with "1" or "+1", and ignores all other non-numeric characters.
- `callback` (optional): a URL where your application can receive an HTTP notification from Veriplace. See [9.5 Getting the Final Result](#) (page 48)
- `callbackContentType`, `callbackFormParamName`: for asynchronous requests only, these parameters control the format of the callback notification. See [9.5 Getting the Final Result](#)².
- `sendUserConfirmation` (optional): true if Veriplace should send a final SMS to the end-user after the user has accepted or declined the invitation. Any other value (or no value) means that Veriplace will skip this final message, in which case the application is responsible for notifying the user of the result on completion.

9.2.1 Common Failure Cases

Because there are privacy considerations sending unsolicited messages, Veriplace may forbid requests under some circumstances or for some applications. In this case, Veriplace will return an HTTP 403 error.

9.3 Successful Requests

On success, you will receive an HTTP 200 ("OK") code and a result document in one of two forms:

- If there is already a Veriplace user with the given mobile number opted-in to your application, the document will contain a Veriplace user ID, and the OAuth access token for locating the user.
- Otherwise, the document will contain a one-time `nonce` (a correlation key) unique to this request, to associate it with its eventual outcome; see [9.5 Getting the Final Result](#) (page 48).

The user ID/access token response is represented in XML or JSON as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
  <invitationResult xmlns="http://veriplace.com/xml/1.2">
    <mobileNumber>mobileNumber</mobileNumber>
    <user id="userID" />
    <token value="token" secret="tokenSecret" />
  </invitationResult>
```

```
{ "invitationResult": {
  "mobileNumber": "mobileNumber",
  "user": { "id": "userID" },
```

```
"token": { "value": "token", "secret": "tokenSecret" }
} }
```

The nonce response is represented in XML or JSON as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
  <invitationResult xmlns="http://veriplace.com/xml/1.2">
    <mobileNumber>mobileNumber</mobileNumber>
    <nonce value="value"/>
  </invitationResult>
```

```
{ "invitationResult": {
  "mobileNumber": "mobileNumber",
  "nonce": { "value": "value" }
} }
```

9.4 Unsuccessful Requests

You will receive an HTTP 400 status code if the request was malformed, an HTTP 401 status code if the provided application-specific access token was invalid, and an HTTP 403 error if the invitation could not be sent for some reason.

Note that Veriplace will return a 400 error if you specify a callback that is not a valid externally accessible HTTP URI, e.g. if its domain is `localhost`.

9.5 Getting the Final Result

If the user responds to the invitation within 24 hours, either agreeing to opt into your application or explicitly declining, then the invitation is complete. To detect this, you can either query the status of the invitation or ask Veriplace to push the result to you.

9.5.1 Querying Results

If you do not specify a `callback` parameter in the invitation, then it is up to you to query Veriplace for the result at some point in the future. You can do this by making an HTTP GET request to the following protected resource URL, using the same `nonce` value that was returned when you made the original request:

```
https://veriplace.com/api/1.2/requests/<nonce>
```

You can repeat the query as many times as necessary, for up to 24 hours; after that, the request expires and any further queries with that `nonce` value will receive an HTTP 404 error.

If the user has not yet responded, Veriplace returns HTTP status 204 ("No Content"). If the user has responded, or if the invitation failed, Veriplace returns a result document similar to the one described above in [9.3 Successful Requests](#) (page 48). It will always contain the `<nonce>` and `<mobileNumber>` elements that identify the invitation, and if

successful it will have the `<user>` and `<token>` elements. If unsuccessful, it will have an `<apiError>` element containing an error code as follows:

- 401, if the user declined the invitation;
- 403, if the invitation turned out to be disallowed after it was submitted.

The format of the `apiError` element in XML or JSON is as follows:

```
<apiError>
  <code>401</code>
  <message>User declined</message>
</apiError>
```

```
"apiError": {
  "code": 401,
  "message": "User declined"
}
```

9.5.2 Receiving Notification by Callback (Push)

If your application can receive HTTP requests, you can tell Veriplace to push a notification to a URL of your choice, which you specify as the `callback` parameter of the original request. Veriplace will make an HTTP POST request to this URL. This message can use one of several formats, depending on what is most convenient for your application:

1. The *summary* format, which is used if you only specified a callback URL, contains a query string in the `application/x-www-form-urlencoded` content type. Since this callback can be delivered non-securely, it does not contain the user's mobile number or access token; you will have to keep track of the mobile number yourself (associating it with the `nonce` value) and use the [Verify Permission API](#) (page 40) to get the token. The query string parameters are as follows:
 - `nonce`: the unique identifier that was returned when you made the original request
 - `code`: the HTTP status code representing the outcome of the request (200 if successful, 401 if declined, 403 if not allowed)
 - `reason`: an optional error description, equivalent to the HTTP status message
 - `user`: the Veriplace user ID, if the user accepted the invitation
2. The *extended* format is used if you also specified a `callbackContentType` parameter, whose value must be either `"text/xml"` or `"application/json"`. In this case, the message will be an `<invitationResult>` object in XML or JSON, containing all the result details if available, as described above. To use this mode, your callback URL must be secure (`https:`).

3. The *form-encoded extended* format is the same as the extended format, but wraps the entire XML or JSON document in a form-encoded parameter -- e.g., `<?xml version=...` becomes `myParamName=%3C%3Fxml+version%3D...`. This format is used if you specify `callbackFormParamName=myParamName` along with `callbackContentType`. This may be useful if your application receives callbacks through a web gateway that can only accept form-encoded content, but you still want the details included in the extended format.

To confirm receipt of the callback, your application must return an HTTP 200 response within 2000 milliseconds. Otherwise, Veriplace will consider the request a failure and will retry up to 3 times before giving up.

Notes

1. http://developer.veriplace.com/devportal/documentation/tutorials/locating_real_users/p4.html
2. http://localhost:8888/devportal/developerguide/invitation-api/edit/093f7d546c228625776f8239785574847f207f29/part-SimpleDocumentContent#devguide_invitation_api_results

10. Using the Polling API

While the [Get Location API](#) (page 24) provides access to location data from one user at a time, sometimes you may simply want to know if there have been recent location updates for any users of your application. This type of batch query is available through the Polling API. Based on the [Simple Update Protocol](#)¹ (SUP) standard which many social network applications use for update feeds, the Polling API returns a list of unique identifiers representing location updates for users that your application has permission to locate; you can then query the location details for any of these identifiers. This is not a true push service, but by making Polling API queries at intervals in the background, you can design your application to use a push model for location updates and avoid querying users who have no new locations.

Note, the Veriplace Java and .NET SDKs provide support classes to further simplify the process so that you can simply designate a listener object to receive location updates. This chapter describes the low-level API in case you are not using these frameworks.

10.1 What Is a Location Update?

The Polling API detects locations that have already been obtained by Veriplace; it does not generate new location requests. Veriplace-enabled smartphones push their locations to the server at regular intervals. Also, for any type of phone, if any application has made a successful on-demand location request, Veriplace will cache that location and add it to the update list. Location updates are therefore equivalent to the locations you would get for any individual user by using [Freedom mode](#) (page 27) with the [Get Location API](#) (page 24), but aggregated from all the users that your application can locate.

10.2 Update Periods

Location updates are cached within a time window that moves forward at fixed intervals. For instance, if the interval is 60 seconds, then a query issued at 12:00:01 might return updates from 11:00:00-12:00:00, and repeating the query would return the same results until 13:00:01, at which point the window would be moved forward by 60 seconds. This is only an example; the actual starting and ending times are determined by the server, and are communicated to your application in the query results, so that you can schedule your next query to be after the last ending time.

As specified by SUP, the results returned by each Polling API query include an "available periods" indicating what intervals are supported by the server. Currently, the default is 60

seconds, but your application can also request an interval of 5 minutes (300 seconds) or 10 minutes (600 seconds).

10.2 Querying the Update List

You can request a list of recent location updates by submitting an HTTP GET to:

```
https://veriplace.com/api/1.2/updates
```

Requests must be signed using HMAC_SHA1 and the **application-specific** Access Token.

To specify a time window other than the default of 60 seconds, add the query string parameter `seconds`, for example:

```
https://veriplace.com/api/1.2/updates?seconds=300
```

10.2.1 Successful Requests

On success, you will receive an HTTP 200 ("OK") code and a document representing the list of updates, if any.

The JSON encoding for this document is described in section 3 of the [SUP specification](#)². It consists of a list of tuples, each of which contains a Veriplace user identifier (see [Using the User Discovery API](#) (page 19)) and a unique identifier representing the user's new location. The document also always has the following properties, whether there were any updates or not:

- `period`: the size of the time window, in seconds
- `since_time`, `updated_time`: the starting and ending point of the time window, respectively, in [RFC3339](#)³ format
- `available_periods`: a list of tuples representing valid intervals you can specify for the time window, each of which contains a number of seconds and the URL you would use for a query with that interval

The corresponding XML encoding looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
  <locationUpdates xmlns="http://veriplace.com/xml/1.2">
    <updatedAt>1970-01-14T23:06:39Z</created>
    <sinceTime>1970-01-14T23:05:39Z</expires>
    <period>60</period>
    <userUpdates id="12345678" <!-- this number is a user identifier -->
      <update id="111222333444"/> <!-- a location update ID for that user
    -->
      <update id="111222333445"/> <!-- another update for the same user
    -->
    </userUpdates>
  </locationUpdates>
```

```

    <userUpdates id="12345679">
      <update id="111222333446"/>  <!-- a location update ID for the
second user -->
    </userUpdates>
    <availablePeriods>
      <period seconds="60"
uri="http://veriplace.com/api/1.2/updates?seconds=60"/>
      <period seconds="300"
uri="http://veriplace.com/api/1.2/updates?seconds=300"/>
      <period seconds="600"
uri="http://veriplace.com/api/1.2/updates?seconds=600"/>
    </availablePeriods>
  </locationUpdates>

```

10.2.2 Unsuccessful Requests

You will receive an HTTP 400 status code if the request was malformed, and an HTTP 401 status code if the provided application-specific access token was invalid.

10.3 Querying Location Details

To obtain the actual location data for any individual location update returned by the update list query above, submit an HTTP GET request to:

```
https://veriplace.com/api/1.2/updates/<updateID>
```

The token for this request must be an Access Token for locating that particular user, rather than the application-specific Access Token used for the update list query.

10.3.1 Successful Requests

On success, you will receive an HTTP 200 ("OK") code and a document describing a single location, using the same XML or JSON encoding provided by the [Get Location API](#) (page 30).

10.3.2 Unsuccessful Requests

A code of 400 indicates that the specified update ID was invalid; a code of 401 indicates that the request did not contain a valid Access Token for locating that user.

Notes

1. <http://simpleupdateprotocol.googlecode.com/svn/trunk/spec/draft-clinton-sup-current.html>
2. <http://simpleupdateprotocol.googlecode.com/svn/trunk/spec/draft-clinton-sup-current.html>
3. <http://tools.ietf.org/html/rfc3339>

11. Using the Verify Locatability API

The Verify Locatability API tests whether a particular mobile number is *potentially* able to be located by your application. Even if the end user has not granted permission to be located, or if a Veriplace account has not yet been created for that number, the Veriplace platform has rules to determine whether those things can happen. You can use that information to tell end users whether your application is compatible with their mobile carrier.

For a launched application (see [2.4 Certifying and Launching Your Application](#) (page 17)), whether a mobile number is locatable depends on its carrier. Applications may be certified for all carriers, or only for specific ones. Veriplace can look up the carrier for any valid North American mobile number and check it against the list of allowed carriers for the application. The API will not tell you what the carrier is, only that it is supported.

For an application that's still in development and hasn't been certified for any carrier, the only numbers that are locatable are the ones you have designated as test phones on the Veriplace Developer Portal site. Also, if and when an end user creates a Veriplace account with that number, the user must choose to enable developer mode or the number will not be addressable by an unpublished application even if it is on the test phone list; see the tutorial on [Enabling Developer Mode](#)¹.

Note that these rules apply to the *number* rather than to the mobile phone itself. Some phones do not have a location feature, or have a service plan that doesn't allow them to be located; if the number has not yet been registered with Veriplace, it's not possible to determine what type of phone the user has until the phone has contacted Veriplace by SMS or mobile web.

Also, locatability is not the same as permission; you will not be able to locate a phone unless the end user has previously granted permission (via web or SMS). See [Using the Verify Permission API](#) (page 40) and [Using the Invitation API](#) (page 46).

11.1 Querying a Mobile Number

To make this request, submit an HTTP GET to the following URL:

```
https://veriplace.com/api/1.2/verify/carrier?mobile=NUMBER
```

The value of the `mobile` query string parameter should be a ten-digit North American mobile number. You may also prefix the number with "1" or "+1", and Veriplace will

ignore all other non-numeric characters, so the following numbers are all equivalent: 5105551212, 15105551212, 1-510-555-1212, +15105551212.

The request must be signed using HMAC-SHA1 and the application-specific Access Token (the one configured for your application as a whole, rather than for a particular user).

11.1.1 Successful Requests

If the mobile number is valid and potentially locatable by your application, you will receive an HTTP 200 ("OK") code. The response has no other content.

11.1.2 Unsuccessful Requests

The following HTTP status codes indicate error conditions:

- 400: the mobile number was not in a valid format
- 401: the application-specific Access Token was not valid
- 403: the mobile number was valid, but the number is not locatable by your application
- 404: Veriplace was not able to determine the carrier for the mobile number (the number may not be in use)

Notes

1. http://developer.veriplace.com/devportal/documentation/tutorials/locating_real_users/p4.html

12. Previous API versions

When new APIs are added to the Veriplace platform, or their parameters or return values are changed, the new versions will use new URLs (`/api/<version>/...`) and the old versions will continue to be supported, with their old behavior, at the old URLs.

The previous chapters of the Developer Guide describe the current API version, 1.2. Here is a summary of the differences from previous API versions.

12.1 API 1.1

12.1.1 Extended callback parameters not supported

For the [Get Location](#) (page 24) and [Invitation](#) (page 46) APIs, the `callbackContentType` and `callbackFormParamName` parameters are not supported prior to API 1.2.

12.1.2 Retrieving asynchronous results

In API 1.1, polling for the result of an asynchronous request with `/api/1.1/requests/<nonce>` does not return a `<getLocationResult>` or `<invitationResult>` object as in API 1.2. Instead, for successfully completed requests, it returns a `<location>` or a `<user>` respectively; for failed requests, it does not return any XML or JSON content but simply an HTTP error code (which, as of API 1.2, is now contained in an `<apiError>` element) with no other information.

12.1.3 Cannot suppress cached position in error results

The `noCachedPosition` parameter is not supported prior to API 1.2; cached position is always included in error results, if available.

12.2 API 1.0

12.2.1 XML namespace

All XML responses from `/api/1.0` URLs use the XML namespace "`http://veriplace.com/xml/1.0`".

12.2.2 JSON encoding not supported for most APIs

In version 1.0, the only APIs that can return a JSON-encoded response are the Polling API and the Invitation API. If you send a request with the header "Accept: application/json" to any other /api/1.0 URL, you will get an HTTP 406 error.

12.2.3 User discovery with multiple results

For requests to /api/1.0/users that return multiple results, the XML encoding is slightly different than in version 1.1. The 1.0 encoding is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<users xmlns="http://veriplace.com/xml/1.0">
  <user id="1378603015410979905" key="1115551212" keyType="mobile" />
  <user id="7799817486972445687" key="1115551213" keyType="mobile" />
</users>
```

12.2.4 Location Responses

In API 1.0, the XML encoding of the <location> element returned by the Get Location API is different from version 1.1 as follows:

- The <uncertainty> element is instead called <accuracy>. (This field was renamed because a higher value indicates a *less* accurate result.)
- The geographic address fields such as <street> and <city> are not enclosed in an <address> element.
- The geographic address elements are always present, even if they have no values.
- The <nearPointOfInterest> element does not exist. To request POI data, you must use API 1.1.

Here is an example of an API 1.0 location response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <location xmlns="http://veriplace.com/xml/1.0" id="1234">
    <created>1970-01-14T23:06:39Z</created>
    <expires>1970-01-14T23:06:39Z</expires>
    <position>
      <longitude>-122.123456</longitude>
      <latitude>37.123456</latitude>
      <accuracy>100.0</accuracy>
      <street>5858 Horton St</street>
      <neighborhood/>
      <city>Emeryville</city>
      <state>CA</state>
      <postal>94608</postal>
      <countryCode>USA</countryCode>
```

```
</position>  
</location>
```

12.2.5 Alternate URL for Verify Permission API

The [Verify Permission API](#) (page 40) can be accessed at `/api/1.0/verify` as well as `/api/1.0/permission`.

A. Veriplace XML Schema

The Veriplace XML schema with annotations is provided below. For schema differences in older APIs, which are still supported, see [Previous API Versions](#) (page 56).

The latest (non-annotated) version can always be obtained from <http://veriplace.com/xml/1.2/veriplace.xsd>¹.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:vp="http://veriplace.com/xml/1.2"
  targetNamespace="http://veriplace.com/xml/1.2"
  version="1.0.1">
```

```
<!-- Top-level elements -->
```

```
<xs:element name="getLocationResult" type="vp:AsyncLocationResultType">
  <xs:annotation>
    <xs:documentation> Result of an asynchronous Get Location API request. <
/xs:documentation>
  </xs:annotation>
</xs:element>
```

```
<xs:element name="identifiedUsers">
  <xs:annotation>
    <xs:documentation> Result of APIs that return a list of users indexed by
their identifying properties. </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="identifiedUser" type="vp:IdentifiedUserType"
maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

<xs:element name="invitationResult" type="vp:InvitationResultType">
  <xs:annotation>
    <xs:documentation> Result of an asynchronous Invitation API request. <
/xs:documentation>
  </xs:annotation>
</xs:element>

```

```

<xs:element name="location" type="vp:LocationResultType">
  <xs:annotation>
    <xs:documentation> Result of APIs that return a single user location or
position error. </xs:documentation>
  </xs:annotation>
</xs:element>

```

```

<xs:element name="nonce" type="vp:NonceType">
  <xs:annotation>
    <xs:documentation> A unique identifier for a request in progress. <
/xs:documentation>
  </xs:annotation>
</xs:element>

```

```

<xs:element name="token" type="vp:TokenType">
  <xs:annotation>
    <xs:documentation> Result of APIs that return a single OAuth access token.
</xs:documentation>
  </xs:annotation>
</xs:element>

```

```

<xs:element name="user" type="vp:UserType">
  <xs:annotation>
    <xs:documentation> Result of APIs that return a single user identifier. <
/xs:documentation>
  </xs:annotation>
</xs:element>

```

```

<xs:element name="users">
  <xs:annotation>
    <xs:documentation> Result of APIs that return a list of user identifiers.
</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="user" type="vp:UserType" maxOccurs="unbounded"
minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

```

```

</xs:complexType>
</xs:element>

```

```

<!-- Complex types -->

```

```

<xs:complexType name="APIErrorType">
  <xs:annotation>
    <xs:documentation> Description of an error result from an asynchronous API
    request. </xs:documentation>
    <xs:documentation> The error code and message correspond to the HTTP
    status code and message that would have been returned if the request had been
    made synchronously. </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="code" type="xs:integer">
      <xs:annotation>
        <xs:documentation> A status code whose meaning is defined by the API
        that generated it. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="message" type="xs:string" minOccurs="0">
      <xs:annotation>
        <xs:documentation> An optional human-readable message to clarify the
        meaning of the error. </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="AsyncLocationResultType">
  <xs:annotation>
    <xs:documentation> Result of an asynchronous Get Location request. <
  /xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="nonce" type="vp:NonceType">
      <xs:annotation>
        <xs:documentation> The unique identifier for the request. <
  /xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="user" type="vp:UserType" minOccurs="0">
      <xs:annotation>
        <xs:documentation> The Veriplace user identifier for the user who was
        located. </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>

```

```

<xs:element name="mobileNumber" type="xs:string">
  <xs:annotation>
    <xs:documentation> The user's mobile number. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="apiError" type="vp:APIErrorType" minOccurs="0">
  <xs:annotation>
    <xs:documentation> Error description, if the request was not valid. <
/xs:documentation>
    <xs:documentation> This means that the request was rejected for
reasons that did not depend on the locatability of the user. </xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="location" type="vp:LocationResultType" minOccurs="0">
  <xs:annotation>
    <xs:documentation> Location information, if the request was valid. <
/xs:documentation>
    <xs:documentation> This may contain a position element for an actual
location, or, if the user could not be located, a positionError element. <
/xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="IdentifiedUserType">
  <xs:annotation>
    <xs:documentation> A mapping between an identifying property (e.g. mobile
number) and a Veriplace user identifier.
  </xs:documentation>
</xs:annotation>
<xs:sequence>
  <xs:element name="user" type="vp:UserType">
    <xs:annotation>
      <xs:documentation> The Veriplace user identifier. </xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
  <xs:attribute name="key" type="xs:string">
    <xs:annotation>
      <xs:documentation> The value that was used to identify the user. <
/xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="keyType" type="vp:UserKeyType">
    <xs:annotation>
      <xs:documentation> The type of value that was used to identify the user.
</xs:documentation>

```

```

    </xs:annotation>
  </xs:attribute>
</xs:complexType>

```

```

<xs:complexType name="IntersectionType">
  <xs:annotation>
    <xs:documentation> Description of a street intersection. <
  /xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="position" type="vp:PositionType">
      <xs:annotation>
        <xs:documentation> Street names, city, state, postal code, and
        longitude and latitude coordinates. </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="InvitationResultType">
  <xs:annotation>
    <xs:documentation> Result of an Invitation API request. </xs:documentation
  >
  </xs:annotation>
  <xs:sequence>
    <xs:element name="nonce" type="vp:NonceType" minOccurs="0">
      <xs:annotation>
        <xs:documentation> The unique identifier for the request. <
  /xs:documentation>
        <xs:documentation> This is omitted if the user was already opted in
        and did not need to be invited. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="mobileNumber" type="xs:string">
      <xs:annotation>
        <xs:documentation> The mobile number to which the invitation was sent.
  </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="user" type="vp:UserType" minOccurs="0">
      <xs:annotation>
        <xs:documentation> The Veriplace user identifier for that mobile
        number, if any. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="apiError" type="vp:APIErrorType" minOccurs="0">
      <xs:annotation>

```

```

        <xs:documentation> Error description, if the request failed. <
/xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="token" type="vp:TokenType" minOccurs="0">
    <xs:annotation>
        <xs:documentation> Permission token for the user, if the request
succeeded. </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="LocationResultType">
    <xs:annotation>
        <xs:documentation> The result of a user location query. </xs:documentation
>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="created" type="xs:dateTime">
            <xs:annotation>
                <xs:documentation> The date and time at which this location was
generated. </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="expires" type="xs:dateTime">
            <xs:annotation>
                <xs:documentation> The date and time after which this location must be
discarded. </xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:choice>
            <xs:element name="position" type="vp:PositionType">
                <xs:annotation>
                    <xs:documentation> Details of the location and optional address
data. </xs:documentation>
                </xs:annotation>
            </xs:element>
            <xs:element name="positionError" type="vp:PositionErrorType">
                <xs:annotation>
                    <xs:documentation> Error description if a position was not
available. </xs:documentation>
                </xs:annotation>
            </xs:element>
        </xs:choice>
        <xs:element name="nearPointOfInterest" type="vp:POIReferenceType"
minOccurs="0">
            <xs:annotation>

```

```

        <xs:documentation> Optional description of a geographic point of
interest near this location. </xs:documentation>
    </xs:annotation>
</xs:element>
    <xs:element name="nearIntersection" type="vp:POIReferenceType"
minOccurs="0">
    <xs:annotation>
        <xs:documentation> Optional description of a street intersection near
this location. </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:unsignedLong">
    <xs:annotation>
        <xs:documentation> A unique identifier for this location result. <
/xs:documentation>
    </xs:annotation>
</xs:attribute>
</xs:complexType>

```

```

<xs:complexType name="NonceType">
    <xs:annotation>
        <xs:documentation> A unique identifier for a request in progress. <
/xs:documentation>
    </xs:annotation>
    <xs:attribute name="value" type="xs:string">
    <xs:annotation>
        <xs:documentation> The unique key. </xs:documentation>
    </xs:annotation>
</xs:attribute>
</xs:complexType>

```

```

<xs:complexType name="POIReferenceType">
    <xs:annotation>
        <xs:documentation> A geographic relation to a street intersection or other
point of interest. </xs:documentation>
    </xs:annotation>
    <xs:sequence>
    <xs:choice>
        <xs:element name="intersection" type="vp:IntersectionType">
            <xs:annotation>
                <xs:documentation> Position and description of an intersection. <
/xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="pointOfInterest" type="vp:POIType">
            <xs:annotation>

```

```

        <xs:documentation> Position and description of a point of interest.
</xs:documentation>
        </xs:annotation>
    </xs:element>
</xs:choice>
<xs:element name="distance" type="xs:double">
    <xs:annotation>
        <xs:documentation> Distance to the position, in meters. <
/xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="heading" type="xs:double">
    <xs:annotation>
        <xs:documentation> Heading to the position, in degrees clockwise from
due north. </xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="POIType">
    <xs:annotation>
        <xs:documentation> Description of a geographic point of interest. <
/xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="name" type="xs:string">
            <xs:annotation>
                <xs:documentation> A descriptive name for the point of interest. <
/xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="category" type="xs:string">
            <xs:annotation>
                <xs:documentation> A general category for the point of interest. <
/xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="position" type="vp:PositionType">
            <xs:annotation>
                <xs:documentation> Position data and optional address data. <
/xs:documentation>
            </xs:annotation>
        </xs:element>
        <xs:element name="phone" type="xs:string">
            <xs:annotation>
                <xs:documentation> The telephone number listed for the point of
interest, if any. </xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>

```

```

    </xs:annotation>
  </xs:element>
</xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="PositionType">
  <xs:annotation>
    <xs:documentation> Geographic coordinates and optional address
information. </xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="longitude" type="xs:double">
      <xs:annotation>
        <xs:documentation> Longitude in degrees. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="latitude" type="xs:double">
      <xs:annotation>
        <xs:documentation> Latitude in degrees. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="uncertainty" type="xs:double" minOccurs="0">
      <xs:annotation>
        <xs:documentation> An uncertainty radius around the longitude and
latitude, measured in meters. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="address" type="vp:AddressType" minOccurs="0">
      <xs:annotation>
        <xs:documentation> Street/neighborhood/city information, if available.
May not be reliable if the uncertainty is high. </xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="PositionErrorType">
  <xs:annotation>
    <xs:documentation> An error returned while determining user location. <
/xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="code" type="vp:PositionErrorCodeType">
      <xs:annotation>
        <xs:documentation> An enumerated error code. </xs:documentation>
      </xs:annotation>
    </xs:element>

```

```

    <xs:element name="message" type="xs:string">
      <xs:annotation>
        <xs:documentation> A context-specific error string. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="cachedPosition" type="vp:PositionType" minOccurs="0">
      <xs:annotation>
        <xs:documentation> The last known position of the user, if available.
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="TokenType">
  <xs:annotation>
    <xs:documentation> An OAuth access token. </xs:documentation>
  </xs:annotation>
  <xs:attribute name="value" type="xs:string">
    <xs:annotation>
      <xs:documentation> The token value. </xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="secret" type="xs:string">
    <xs:annotation>
      <xs:documentation> The token secret. </xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>

```

```

<xs:complexType name="UserType">
  <xs:annotation>
    <xs:documentation> A Veriplace user identifier. </xs:documentation>
  </xs:annotation>
  <xs:attribute name="id" type="xs:unsignedLong">
    <xs:annotation>
      <xs:documentation> The unique key for this user. </xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>

```

```

<!-- Enumerated types -->

```

```

<xs:simpleType name="PositionErrorCodeType">
  <xs:annotation>

```

```

    <xs:documentation> Enumeration of possible error codes returned for user
location. </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:integer">
    <xs:enumeration value="0">
      <xs:annotation>
        <xs:documentation> OK </xs:documentation>
        <xs:documentation> Position determination was successful. <
/xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="100">
      <xs:annotation>
        <xs:documentation> Position Failure </xs:documentation>
        <xs:documentation> Position determination was not successful. <
/xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="110">
      <xs:annotation>
        <xs:documentation> Position Determination Temporarily Unavailable <
/xs:documentation>
        <xs:documentation> Position determination was not successful and the
underlying position determination technology appears to be temporarily unable to
obtain usable data. </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="200">
      <xs:annotation>
        <xs:documentation> Restricted </xs:documentation>
        <xs:documentation> Position determination was not attempted for
privacy reasons. </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
  </xs:restriction>
</xs:simpleType>

```

```

<xs:simpleType name="UserKeyType">
  <xs:annotation>
    <xs:documentation> Enumeration of the properties that can be used to
identify a user. </xs:documentation>
    <xs:documentation> These are used for the keyType attribute of an
identifiedUser element. </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="mobile">
      <xs:annotation>
        <xs:documentation> The key is a mobile number. </xs:documentation>

```

```
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value="email">
    <xs:annotation>
      <xs:documentation> The key is an email address. </xs:documentation>
    </xs:annotation>
  </xs:enumeration>
  <xs:enumeration value="openid">
    <xs:annotation>
      <xs:documentation> The key is an OpenID identifier. </xs:documentation
>
    </xs:annotation>
  </xs:enumeration>
</xs:restriction>
</xs:simpleType>

</xs:schema>
```

Notes

1. <http://veriplace.com/xml/1.0/veriplace.xsd>