Developer Manual

Ver. 0.4

HERE & NOW

GPS – Compass, Realtime, Share, Explore, Search, Reality

REALITY – Impressions, Tracks, Geo-Tagging, Realtime, People

PEOPLE – Anonymous, Communities, Family, Friends, Dating, Communication

COMMUNICATION – Talk, Inbox, Comments, Invitations, Websites

WEBSITES – Mashup, Login, Tokens, Community, Mobile, Real-World

Contents

1 Introduction

Welcome to the wonderful world of GpsNose :-)!

GpsNose © Nemanicnedanic, Inc. is a completely free, anonymous and a *real-world* platform, in contrast to all the "virtual social" platforms out there. You can find real new people and explore real places around you in real-time.

This manual is meant for the *developers* and not for the end-users, which can find the user-guide here: http://www.gpsnose.com/home/doc

2 Geo-enabling your application

2.1 Motivation for real-world context

Today's apps, be it a website/rich-client/mobile, are mostly *reality-disconnected*. You let your users to sign-in into your app, so they can enter and read data. At the end, they exit the running app and are *gone*. Your users "live" only in the cyber-space and the data they push/pull from your platform are accessible by the business-logic, which has mostly nothing to do with the reality around your users.

The apps lack the reality-context. The users from the same community can't find each other in their area, they are offline when not currently visiting your website, you can't show them the relevant data in their current nearby area etc. When your app needs the real user's area, like some real estate platform or dating, you are forced to use some local workarounds, like some ZIP database in a given state, which is country-specific and never accurate.

The GpsNose platform, www.gpsnose.com, enables you to integrate the reality around your users into your software, using the free open-source SDK libraries, C# or PHP, available here: https://wiki.gpsnose.com/download/

Your users stay "alive" also when they leave your app, as they are using the GpsNose mobile app.

2.2 GpsNose integration overview

GpsNose is the backbone for your application's geo-scope. You take the open source SDK (as is, or port it) and connect your existing/new app to the GpsNose backbone.

GpsNose backend gives you these services:

- User management: authentication and authorization, lost passwords, communities etc.
- Nearby stuff: geo-based community data, like other users, places, tracks, images, events etc.
- News: time-based community data, like creation of places, tracks, new members, updates etc.
- Tokens: extension-point for your custom use-cases – create, scan and get QR-based tokens

2.3 What do you need

To integrate the real-world data context into your application, you write code with the provided GpsNose SDK and your users use the free GpsNose mobile app. The bridging is done by registering your website as a GpsNose *mashup*.

2.3.1 Community website

You need a website, which is under your control. Your users will join your new *community*, which is then called the same as your website's URL is.

The application you are integrating, can live of course outside such website, i.e. you can provide your geo-scoped application logic directly at such website, or you can optionally give your users a mobile app or rich-client. But you need a website in any case, which identifies your community.

The website doesn't have to implement any fancy logic; it's *just there*. It serves:

- Your users as an identifiable point, where they can reach some information, like help or other community members etc. and it serves also:
- You for bridging the GpsNose backbone into your SW solution; GpsNose calls such website a *mashup* and it requires you to validate once the ownership of such *mashup* website.

EXAMPLE: You plan to develop a geo-scoped real-world game *GeoHamster.* The first thing you want to do is register a domain for your planned community, such as www.geohamster.com.

2.3.2 Mashup connection

To tell GpsNose about your planned community, you create a *mashup* connection between the GpsNose backbone and your application.

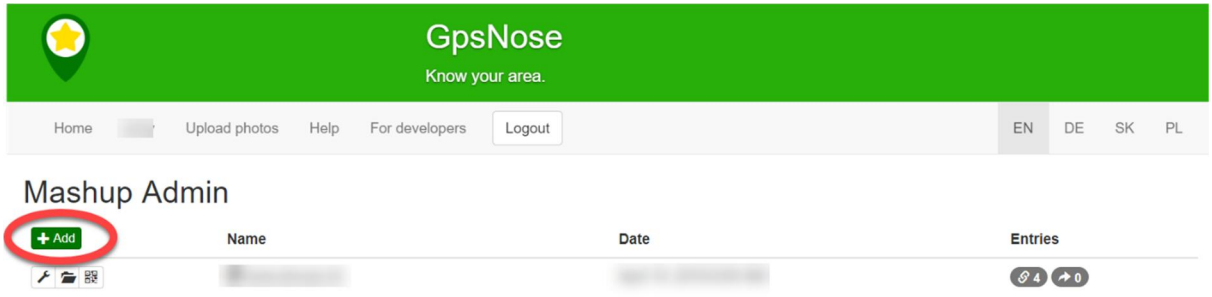
This must be done only once. You define the mashups:

- Directly at the GpsNose website, or
- Through the open source SDK using the GnAdminApi class, or
- Comfortably by a CMS module, such as TYPO3.

See the *Links* chapter for getting the coordinates.

EXAMPLE: For your planned *GeoHamster* game, you already have registered the domain geohamster.com. Now you need to tell GpsNose about it, so you create the *mashup* connection following these steps:

1. Visit the Mashup-Admin page at http://www.gpsnose.com/Developer/MashupAdmin and use the "Add" button:

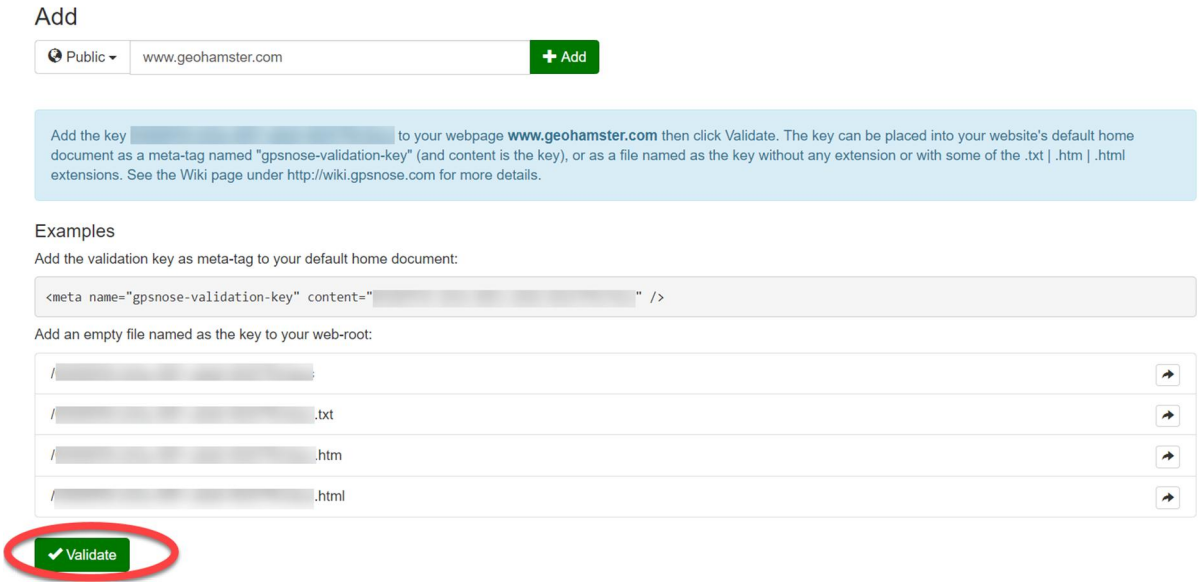


2. Enter the www.geohamster.com as the mashup community name.

As you want *anybody* can join your community without needing an invitation, you leave the "Public" option selected. You could of course create a *closed* or *private* community here (see the GpsNose user manual for more information about the community access-levels).
The access-level decision is not reversible, so think twice before proceeding. Later, you can create more sub-communities, with different access-levels, for example some game-progress communities like "level1", "level2" etc., which could be created as *private* sub-communities.

NOTICE: the community-membership (i.e. the communities your users join) and the data your community members create, is later *visible to everybody* in the GpsNose. When you want the data be invisible for the non-members, you must create the respective community as *private*.

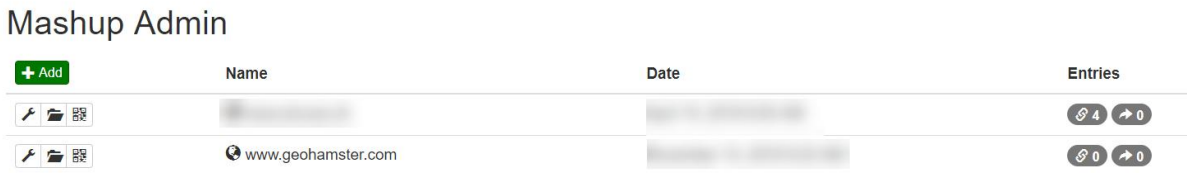
3. Validate the ownership of geohamster.com domain



To tell GpsNose the community name and website www.geohamster.com is really under your control, you must *validate* it. This can be done by putting the random validation-key under your root web folder, with or without an extension (see the screenshot above), or by putting the meta-tag header into your default home document.

When you are using a CMS module or the direct SDK calls on the GnAdminApi, you can automate this process – you don't need to put the random validation key yourself into the target website as a key-file or a meta-tag: this is done by the CMS automatically (or by you, coding with the SDK).

NOTICE: As long as you do not validate the mashup site, the corresponding community is not created. After the successful validation, the community is created, such as the "www.geohamster.com" community and your users can join it as community members. You can see all the successfully validated mashups in the Mashup-Admin page:



2.3.3 SDK (or CMS)

To call the GpsNose backend services, you can use the SDK provided here:

https://wiki.gpsnose.com/download/

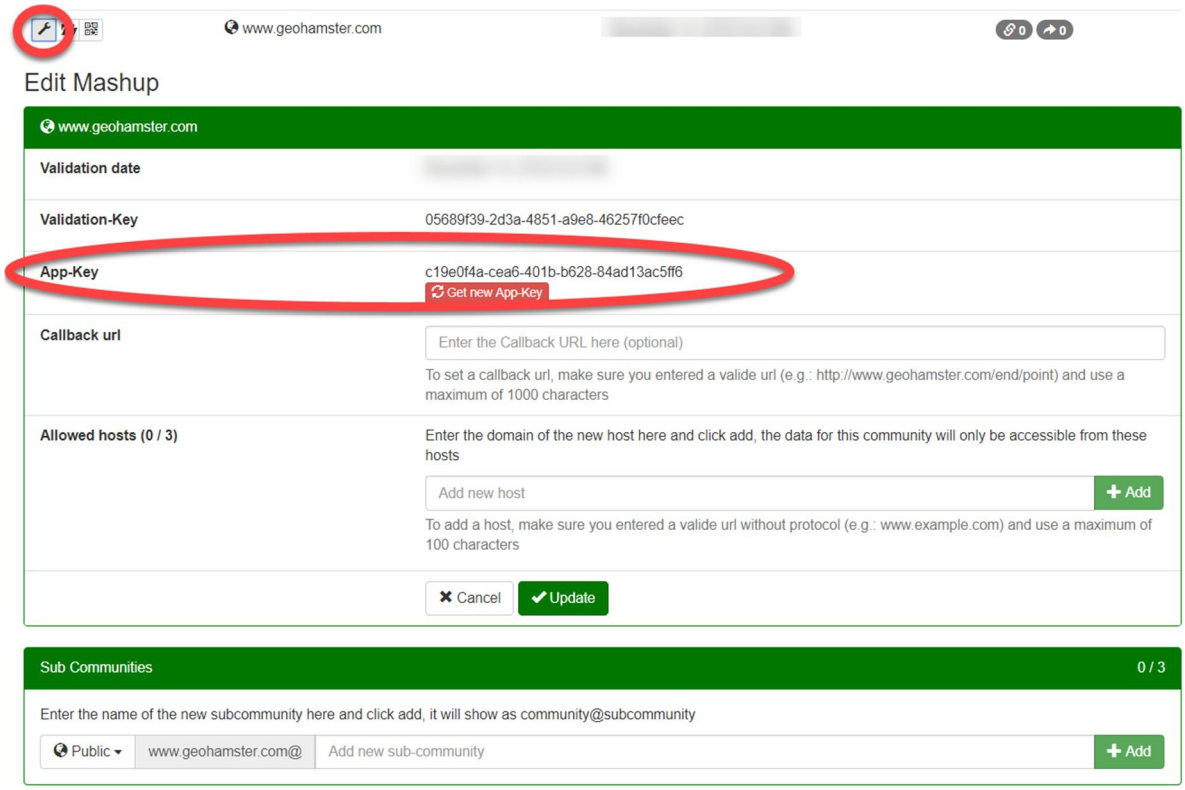
There is a C# and PHP reference implementation, which you can port to any other environment ;-). We can support you doing so and will place a link to your project under our Wiki page.

The SDK is available both as open source code and as packages (both NuGet for .NET and Composer for PHP).

NOTICE: As an alternative to coding with the SDK, a ready-made CMS module is a comfortable way for integrating the geo-scoped services into your app, which is not the scope of this document.

2.3.4 App-Key

The first thing to you need when calling the GpsNose SDK is the *app-key* you've got after a successful validation of your mashup:



Normally, you don't want to give your app-key away to the public, as your community and user data would be compromised. Should the app-key get disclosed, you can regenerate it with the "Get new App-Key" button seen above.

The app-key belongs onto your server and only your server should be able to call the API directly. In the case you want to develop a mobile app or rich-client with accessing the GpsNose platform, you can call first your own webserver service, which then calls the GpsNose backbone. In this manner, you have under control your API calling quotas and you can use the SDK internal caching on your webserver.

2.3.5 Your custom code

The geo-scoped backend-services of the GpsNose platform are accessible as stateless JSON services. These are wrapped into the provided SDK, so you can focus on coding with classes/methods, instead of dealing with JSON streams and networking.

The SDK implements already the caching and paging mechanism, so it's easy to code with it.

2.4 What your users need

2.4.1 Mobile app GpsNose

Your users need to install and use the mobile app GpsNose, which serves as the connection to the GpsNose platform. The mobile app is a free flexible tool staying online in their current area. Your user can quit your custom application, like closing the browser, or exiting a rich-client – but their mobile app GpsNose keeps running and they keep connected within their neighborhood.

2.4.2 Join your community

To have a user be part of your mashup community, he must *join* your community. There are few ways a potential new member can join:

- By checking the profile of somebody around, he sees his communities and can joint them right from his profile
- By scanning a QR-code from your website or from a printout (such QR-code is generatable by the SDK)
- By accepting a membership invitation from somebody

In any case, the user becomes a member of your mashup community and you can push/pull him geo-relevant data when he opens your app.

3 Coding with the GpsNose SDK

In GpsNose, there are many data items, which can be created and explored by your community-members.

Most of the functionality your members have in their GpsNose mobile app is also accessible to you in the SDK, so you can integrate the real-world data from your community members into your custom app, as long as the data-items were marked with your mashup-community.

3.1 General concepts

3.1.1 GnApi

The API is object-oriented and the very first class to get to all the API calls is always the *GnApi*:

```
var api = new GnApi();
```

The *GnApi* instance can be reused as many times as you want. Having the *GnApi* object, you need to decide as *who* you would like to call the API:

- As the *end-user*: your GpsNose SDK-calls on your website run impersonated as the user, which logged into your website with the QR-code (or from the mobile APP with the mobile-browser), or
- As the *admin-user*: the SDK-calls run as you, the mashup-administrator of your website.

For the administrative tasks, like resetting the mashup-site's app-key, or adding sub-communities, you can use of course the developer's console available at the GpsNose website here, but you might want to automate something, like adding and verifying a new mashup site for your account, so all such tasks can be accomplished also using the admin-API calls.

On the other hand, the end-user can login into your website using his GpsNose mobile-app, scanning the QR-code (will be explained later) from your website. Or, he can login directly from his GpsNose mobile-app, from within the community-details window: he is then redirected straight to your website on his mobile phone (i.e. using the mobile's web-browser). Then your website can enter/read the data in his name, so he sees other users and things in his real nearby area, like impressions, tracks, events etc.

The end-user can also call some services as a guest, so the corresponding sink object *GnLoginApiEndUser* can be in a not-logged-in state.

Both the GnApi.*GetLoginApiForAdmin()* and *GetLoginApiForEndUser()* work similarly. When called, you can provide your own loginId for identifying the login-session, which is simply an alphanumerical random string, at least 10 chars long, or you can leave it empty, to get a new Guid string from the SDK. It's just a session-id, which identifies globally (worldwide, not just within your website) a user, after he logs-in.

The *GnLoginApiBase* objects – *GnLoginApiAdmin* and *GnLoginApiEndUser* – give you the needed API services.

3.1.2 Paging and Ticks

A lot of API calls use the time-based paging, like in the *News* example. The data-page you get contains a data-segment from a given time-point, let's say 20 items in the page. Then, you can page-through to the next-page supplying the API the *lastKnownTicks* parameter, which is some item's creation ticks you already have received before.

The meaning of "last known" should be self-explanatory in the concrete use-case scope. There are use-case, where the items are listed in the chronological creation-order (smaller -> bigger, like in the *Tokens*) and other use-cases have the items saved in the upside-down order (bigger -> smaller, like in the *News*), so you get the newest data on the beginning.

The *Ticks* value represents the number of 100-nanosecond (i.e. one ten-millionth of a second) intervals that have elapsed since 0:00:00 UTC on January 1, 0001, in the Gregorian calendar.

Everywhere you get or set any kind of *ticks*, they contain the UTC date-time.

The *pageSize* is available in all the API calls which involve paging. When you leave the parameter out, the default will be used. When you set it bigger than the allowed maximum, the maximum will be taken instead.

3.1.3 Caching and Quotas

Most of the SDK API-calls are cache-enabled: if the same API-request parameters have already been used a short time ago (default is 15 minutes), the response is read from the SDK's cache, instead of contacting the GpsNose server.

The advantage of this is:

- Saving time/traffic costs needed for contacting the external web service and
- Saving your monthly calling quotas on the GpsNose server side.

As GpsNose is free for anybody (well, as long as we can pay for it ourselves), we must ensure websites with high traffic won't make us bankrupt ;-). There are calling-quotas applied for your website, which can be seen and monitored in the Mashup admin developer console (see *Links* chapter).

Should you need to explicitly clear the cache content, you can call:

```
GnCache.ClearCache();
```

3.1.4 Local Settings

The CMS or web modules dependent on the SDK must know a few defaults you are using. Although these defaults are used by the CMS or web modules, the *GnLocalSettings* are located in the main SDK, as otherwise every supported web or CMS technology would need to copy/paste these for own use.

You can set the default in your app's startup code like this:

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    RouteConfig.RegisterRoutes(RouteTable.Routes);

    // set as you need
    GnLocalSettings.GnImagesPath = "/Content/GpsNose/Images";
    GnLocalSettings.AppKey = System.Environment.GetEnvironmentVariable("Gn_AppKey");
    GnLocalSettings.Community = System.Environment.GetEnvironmentVariable("Gn_Comm");
}
```

3.2 Community

Users can see each other in their neighborhood, can use the area-chat for making new real social-connections, can use the P2P internal messaging, can enable location-sharing etc.

You can read all the community members (or you can get those nearby your current user, see later).

3.2.1 Getting all community members

```
public void Wiki_AllCommunityMembers()
{
    // get the GN platform api
    var api = new GnApi();

    // get the end-user login-api in a not-logged-in state in this example
    var loginApi = api.GetLoginApiForEndUser(_site.AppKey);

    // list the community members
    var communityApi = loginApi.GetCommunityApi();
    var members = communityApi.GetMembersPage();
    while (members.Count > 0)
    {
        // the members are key-value pairs [loginName, joinTicks]
        // use oldest known member-join to get the older subscriptions
        members = communityApi.GetMembersPage(members.Min(m => m.Value));
    }
}
```

NOTICE:

- You could specify an optional sub-community and/or page-size in the `GetMembersPage()`
- This call is available also for the guest user, as the caller's location is not needed; your mashup site must have the *ListMembers* ACL flag set within the mobile app (managing the community ACLs is described in the GpsNose user guide – see the *Links* chapter)

3.2.2 Getting the community entity

It's practical to have access to the community entity itself, so you can adapt your UI dynamically, like:

- Showing or hiding the members-listing based on the *CanListMembers*, or
- Enabling admin-buttons for the community-admin user etc.

```
var communityApi = loginApi.GetCommunityApi();
var comm = communityApi.GetCommunity();
```

When you leave out the sub/community parameter of `GetCommunity()`, you will get your main community instance, known by the API's app-key. In this way, you can get to your community just by knowing your app-key.

3.2.3 Generating QR-code to join

Your app can generate a QR-code, which enables your potential member to join your community. Such QR-code can be printed out and placed at the table in a restaurant, can be sent as a post-card, or displayed at your website.

```
var qrJoinComm = communityApi.GenerateQrCodeForCommunityJoin();
```

The returned byte array is the QR-code PNG image, which you can save, display or print.

3.2.4 Inviting concrete user to join

You can invite a concrete user to join your sub/community. He will get your message as an internal GpsNose post and can directly accept the community invitation.

This is practical especially when you forbid the members inviting new members (ACL: *MembersInviteMembers*) or when the community is closed/private and you want to promote the user to new communities based on your custom business-rules.

```
communityApi.InviteMemberToCommunity("dracula");
```

As always, you can specify the community as your main or sub-community, or leave out the *community* parameter to use your default main community.

NOTICE: When you're calling this as an anonymous user, the invitation is sent on *your* behalf, i.e. the community creator.

3.3 Nearby

To be able to show the user the data relevant in his current area, he must be logged-in, using the QR-code you generated and showed him.

3.3.1 Members Around

```
public void Wiki_MembersAround()
{
    // get the GN platform api
    var api = new GnApi();

    // get the end-user login-api, which must first log-in in this example
    var loginApi = api.GetLoginApiForEndUser(_site.AppKey);

    // generate the QR-code for login
    var qrCode = loginApi.GenerateQrCode();

    // TODO: here, you must show the user the QR-code so he can scan it

    // set max wait-time as 1 minute for example
    var cancelSource = new CancellationTokenSource(1 * 60 * 1000);

    // wait for the end-user to log-in with scanning the QR-code
    var task = loginApi.WaitForLogin(cancelSource.Token);
    try
    {
        task.Wait();
    }
    catch (OperationCanceledException)
    {
        // cancelled by time-out or user-cancelled: do nothing..
        throw new Exception("user didn't log-in");
    }

    // list the community members around
    var nearbyApi = loginApi.GetNearbyApi();
    var members = nearbyApi.GetNosesAround();
}
```

3.3.2 Places

Places are points of interest generated by your community, which can be private or public and always hold a precise location and an image.

```
var places = nearbyApi.GetPoIsAround();
```

3.3.3 Impressions

An *Impression* is either a pictures and/or text, which is bound to a precise or an approximate location.

```
var impressions = nearbyApi.GetImpressionsAround();
```

3.3.4 Tracks

A *Track* is a manageable set of recorded waypoints when hiking, driving a car etc., optionally enriched with Places and/or Impressions, able to geo-tag the user's pictures made with an external camera without GPS data.

```
var tracks = nearbyApi.GetTracksAround();
```

3.3.5 Events

Your members can organize *Events*, invite other members, vote for the best event-date, confirm a selected date or subscribe to multiple-dates, with the full time-zone support and notifications.

```
var events = nearbyApi.GetEventsAround();
```

3.4 News

The *News* keep track of what is going on in your community.

```
public void Wiki_PageThroughAllNews()
{
    // get the GN platform api
    var api = new GnApi();

    // get the end-user login-api, which stays not-logged-in in this example
    var loginApi = api.GetLoginApiForEndUser(_site.AppKey);

    // get the news-api, which is an guest-enabled api
    var newsApi = loginApi.GetNewsApi();

    // get the news
    var news = newsApi.GetNewsPage();

    // continue reading, until more news-data keeps coming
    while (news.Count > 0)
        news = newsApi.GetNewsPage(lastKnownTicks: news[0].CreationTicks);
}
```

NOTICE

- The news reading is user sensitive: when the calling user is anonymous, he gets only the news visible to the anonymous public.

- News with repeating content, like editing the same item many times in a short time-frame, are already filtered out for you, so you don't have to do it yourself.

3.5 Comments

Your community can comment any existing items, like tracks or places, or it can comment the community itself, creating the *community comments.*

When commenting an item, the original item's creator is notified by the internal GpsNose post on his mobile and all the commented item's participants are notified as well. The item's creator can delete such comment if he doesn't like it.

The *community comments* give you the possibility to implement some kind of community forum, which could have the form of a blog, or announces, or anything you need. You can show the comments of the main or sub-community and you can let your users enter new or edit an existing comment. You can also disallow your users to enter new comments into a given community forum (ACL: *CommentsFromMembers*); then only the community admins can write the community announcements.

The comments are visible also in mobile app, in the item's or community detail window – and when entered from the mobile app, they are shown also at your website.

When the comments are entered, there are also the news generated, so you can decide, what to show where and how.

3.5.1 Reading the comments

To read the comments, your user doesn't have to log-in, as long as the comments come from a *public* (and not *closed* nor *private*) community.

```
var commentsApi = loginApi.GetCommentsApi();
var comments = commentsApi.GetCommentsPage();
```

When you leave out the *itemType* and *itemId*, you are referring to the community comments by default. Otherwise, you can specify for example an impression-type, with some existing impression-id, for reading its comments.

3.5.2 Entering and deleting the comments

The commenting user must be logged-in to be able to modify the comments.

```
// create a comment
long commentCreationTicks = commentsApi.AddComment("Hello, this is a demo!");

// edit the comment
commentsApi.UpdateComment(commentCreationTicks, "Hello, this is an updated comment.");

// delete the comment
commentsApi.UpdateComment(commentCreationTicks, null);
```

3.6 Mails

Using the SDK, you can both read and write GpsNose internal mail messages. This can be used as notifications-framework for your workflows, or implementing community-chat etc.

3.6.1 Reading mails

The mails can be read *only* for the community-creator account. Otherwise, it would be a security problem of course ;-). It doesn't matter, if your current *GnLoginApiEndUser* instances is logged-in by anybody or is in a guest state: you always will get *your* mashup-creator's mails. Be careful not to let a user log-in and thinking "Oh, now let's receive and show him his mails!", as he gets *yours* instead.

```
public void Wiki_GetAllMailForCommunityCreator()
{
    var api = new GnApi();

    // the loginApi stays in an anonymous-caller state in this sample
    // notice: also when somebody logs-in, ONLY community-creator's mails are read
    var loginApi = api.GetLoginApiForEndUser(_site.AppKey);

    // read the mails for the community-creator
    var mailsApi = loginApi.GetMailsApi();
    var mails = mailsApi.GetMailsPage();
    while (mails.Count > 0)
        mails = mailsApi.GetMailsPage(mails[mails.Count - 1].CreationTicks);
}
```

The mail-reading executed done by using the SDK, does *not* reset the GpsNose internal flags like "HasNewMails" and "LastMailChecked" and such. This means, that your mobile app still reacts on any new mails as being really new, although your mashup's backend might have already read your messages, because of some automation logic.

NOTICE

Why is the `GetMailsPage()` implemented in the end-user login-scope, although you always get *your* own mails? Because it's practical: you have already the end-user login-API instance at hand, for example after sending emails on your user's behalf. The user logs-in and triggers some mail-generation, him as the sender.

Now, in the same moment, you might need to get the community-creator emails, for some mail-communication automation, which must work also without being logged-in of course. The end-user based API seems to be the most logical place where to receive the mails from; otherwise, you would need to create another login-API instance, which would give you still your own emails anyway, independent of the logged-in state it's based on.

The other possibility would be to place the *GetMailsPage()* call outside of the *GnMailsApi,* say to your admin-API. But then again, the *GnMailsApi* would lack this mail-reading functionality, which would be quite confusing. That's why the *GetMailsPage()* is in the *GnMailsApi* and is based on any end-user API instance, although it's not reading the logged-in user's emails.

3.6.2 Sending mails

You can send a message to either:

1. One concrete user, or
    - The target user *must* be at least in one of your mashup's sub- or main-community
    - It's *not* allowed sending a message to a group of users, as it's allowed in the mobile app

2. The whole community
    - You can specify your mashup's main-community or sub-community, e.g. "%www.geohamster.com" or "%www.geohamster.com@level-1"

When the login-API instance is *not* logged-in yet (i.e. it's guest-called), the sending user is default you, i.e. the community creator.

```
public void Wiki_SendMail()
{
    var api = new GnApi();

    // when nobody logs-in, the community-creator is the sender
    var loginApi = api.GetLoginApiForEndUser(_site.AppKey);

    // send a mail as community-creator
    var mailsApi = loginApi.GetMailsApi();

    // to concrete receiver: MUST be in some of mashup's main/sub-communities
    mailsApi.SendMail("helmut", "Hello to helmut");

    // to whole community: can be main- or sub-community of your mashup
    mailsApi.SendMail("%www.geohamster.com@level-1", "Hello to sub-community!");

    // now, send a mail as some logged-in user
    var qrForLogin = loginApi.GenerateQrCode();
    // TODO: show the QR-code for scanning the user signing-in
    loginApi.WaitForLogin().Wait();
    var mailsApiSignedId = loginApi.GetMailsApi();
    mailsApiSignedId.SendMail("guest", "Hello from SDK!");
}
```

3.7 Mashup Tokens

The *Mashup Tokens* are special in GpsNose, as they are designed only for your own custom usage and not for the mobile app GpsNose itself. It's a kind of an *extension point*, where only you can decide, who/what/when/where/how can deal with them.

The platform doesn't save the token entity you freshly generated: the tokens are persisted only when scanned-in by your users. It's like class and object: generating the token is the *class* and scanning it in is the *object.* You can't list all the tokens you ever generated, but you can list the tokens which were scanned-in somewhere sometime by somebody. You need to print or store the generated token yourself, so you users can scan it.

NOTICE: A CMS-plugin based on this SDK *can* implement the token persistence, so it's definition (i.e. the "class", before even scanning it) persistence itself for user's convenience.

When generating the token, you can set a few optional parameters:

- *string payload*: any custom string for your needs – when scanned, this data is available to you, so you know *what* was scanned; the other metadata (*who, when, where* etc.) is available to you automatically based on the scanning event.
- *long validToTicks:* on which date will the token expiry, i.e. until when can be scanned this token
- *float valuePerUnit:* in the case your users can scan a kind of "order" (like "I order this pizza"), you can define a value per unit; this information should be later newly calculated and by the receiving mashup site and confirmed by your user, as the price could have changed probably in the meantime

- *string label:* the visible label in the app, when scanning the token
- *GnMashupTokenOptions options:* defines the app's behavior and UI controls while scanning the token – it's a bit-field, so you can combine the options:
    - *NoOptions = 0*
    - *BatchScanning = 1:* the app will wait, until the user scans more tokens and completes a kind of "order" (behaves like a shopping cart)
    - *CanSelectAmount = 2:* the user can set the amount for every item being scanned
    - *CanComment = 4:* the user can set an additional comment for the scanned item (like "this pizza without peperoni")
    - *AskGpsSharing = 8:* the user will be asked, if he wants to allow GPS-sharing while the token is being processed (e.g. the pizza-courier can see the ordering-user's location and vice versa, until the pizza is being delivered)

EXAMPLE: Developing the *GeoHamster* game, you need the *Hamster-Box* holding the treasure be scannable by the *Hamster* player who found it, so your game knows the cache was exposed by the player.

You can first use the SDK to create a *Mashup Token* with any data you need inside, like the treasure-ID and tell the user hiding the box to print the generated QR-token. The token printout is then glued onto the box and hidden in the woods.

As the other user finds it, he scans the printed token by his GpsNose mobile app (which can be made offline as well) and your website gets notified about it: who/when/where/what was scanned. Your mashup website can define a *Callback-URL*, which will be called synchronously, when your user scans the token (5 seconds maximum for your website to return, or it's marked as time-out, although your website callback might later have completed the call).

The scanned-in tokens are available to you for pulling and processing also asynchronously, using the *GnMashupTokensApi*.

NOTICE: Be careful not to expose your mashup's app-key, as the API GetMashupTokensPage() needs only the app-key for reading-in the scanned tokens. It's your responsibility the data stays protected with the app-key, as it contains the login-names with the geo-location and time of the scanned tokens.

3.7.1 Creating a token

When creating a token, keep in mind these key-points:

- The token you create is to be persisted or shown by yourself: it's not persisted in GpsNose while not scanned-in by a user.
- The token can be used just once or reused over and over again, when being-scanned by your users.
- The token can stay valid for scanning-in forever or you can let it expiry after a given UTC ticks time. For example, you glued it somewhere outside and you want to set its time-to-live frame, so nobody can scan it past that point by the GpsNose mobile app.
- The token holds some data publicly readable and some data encrypted
    - Public data: creator, creation, validto, options, enc, ver, valperu, label
    - Encrypted data: payload

- The token holds your custom *payload* encrypted, so nobody can read any internal data from the QR-code itself; the payload gets decrypted in the scanned-token event while sending to your mashup site.
- The token cannot be manipulated (i.e. somebody would like to create some non-existing pizzas for you, defining and printing a similar QR-code), as the tokens holds an encrypted checksum of all its fields.

You can generate a token using the API like this:

```
var tokensApi = loginApi.GetMashupTokensApi();
var qrToken = tokensApi.GenerateQrTokenForMashup("myId123");
```

The qrToken is a byte[] array with the QR-code PNG image. Here is the API interface:

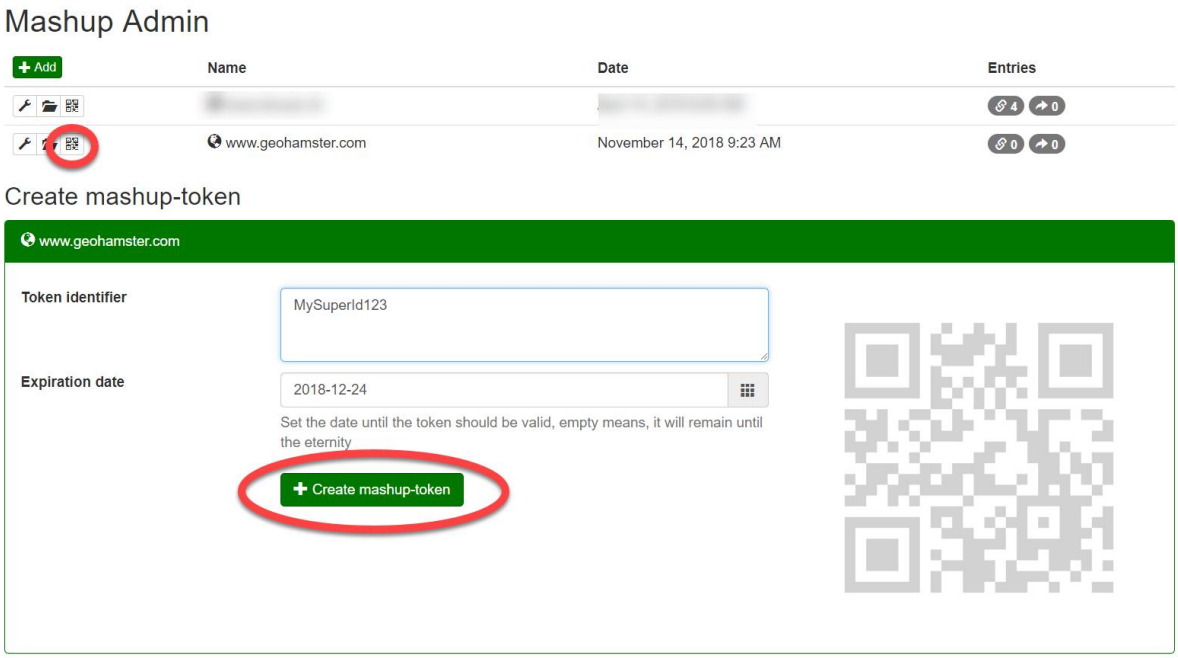
```
public byte[] GenerateQrTokenForMashup(
    string payload,
    long validToTicks = 0,
    float valuePerUnit = 0,
    string label = null,
    GnMashupTokenOptions options = GnMashupTokenOptions.NoOptions)
```

When creating the token, you must specify at minimum some *payload* data for your use-case scenario.

There are also additional fields you can specify:

- *validToTicks*: when specified, you can define the time-to-live for a token; let's say you are creating some product-tokens, for scanning-in pizzas: most probably you wouldn't like your customer to scan-and-order a pizza-token, which was created 10 years ago ;-)!
- *valuePerUnit*: every token can hold an optional value; it's for order-like use-cases, so your scanning user can see the value/price for the scanned-in token, like when ordering a "Large pizza Margherita with chees"
- *label*: this label is displayed to your scanning user, so he knows, what was scanned. This field can optionally hold a JSon-data with additional sub-fields, which are UI specific:
    - *sub:* this can be a mashup-subsite, so you can target a subsite of your mashup, like "level-1" in your game "www.geohamster.com" mashup-site, or "pizza-eaters" in your www.yourcoolfoodorderingsite.com
    - *img:* when defined, the scanned-in token is displayed with an icon in the mobile-app
    - *label:* this is the actual label, which is displayed; when the parent "label" field wouldn't hold JSon, it has the same meaning
    - EXAMPLE
      {"sub": "pizza-eaters",
      "img": "http://www.yourcoolpizzastore.net/img/pizza1.jpg",
      "label":"Pizza Margherita XXL 55cm"}

Alternatively, you can generate the token also in the Mashup Admin page and save/print it from there:



3.7.2 Scanning the token

The scanning is done using the mobile app GpsNose your users have. All the scanned tokens are stored in the GpsNose platform for 14 days so you have plenty of time to pick up.

3.7.3 Be notified while scanning: sync

When you have defined the *Callback URL* in the *Mashup-Admin* page, you will get the notification immediately when the user scans your QR-token. Be careful to complete your call within 5 seconds (calling your site including) as otherwise the token will be marked as time-out and the user will be notified your website doesn't answer, regardless if you could finish successfully the callback just a few moments later.

All the tokens are available for you to pick up, regardless if they were scanned with or without the defined *Callback URL* – and regardless if the callback was successful or not.

The rules for interpreting what was going in the token-scanning process, based on the content of the *GnMashupToken* instances you get when reading the scanned tokens:

CallbackResponseHttpCode	Meaning
0	The token was scanned and nobody tried to call your mashup's *Callback URL*, as it was not defined in the time of token-scanning.
200	The mobile app GpsNose informed the user, that your server was able to handle the token-scanning successfully and shows optionally the *CallbackResponseMessage* when defined.
!=200	The mobile app GpsNose showed the user a general message, that your server was not able to handle the token-scanning. You can check the technical details in the *CallbackResponseMessage* reading-in the tokens.

When you return a (short!) message from the callback function, in the HTTP stream as a string, this is preserved in the *CallbackResponseMessage* and shown to the scanning user in the GpsNose mobile app, but only if the CallbackResponseHttpCode code == 200, so you can optionally tell the user something, like "Congratulation, you found the treasure!" or "You can't send this item, it's locked by user X".

The expected interface for your callback:

```
[HttpPost]
public void Post(
    // token-definition
    string creator,
    long createdTicks,
    string payload,
    long validUntilTicks,
    string label,
    decimal valuePerUnit,
    int options,
    // scan-event
    string user,
    long scannedTicks,
    double lat, double lon,
    long recordedTicks,
    // user-supplied
    int amount,
    string comment,
    bool isBatchCompleted,
    bool isGpsSharingWanted,
    long batchCreationTicks)
```

The parameters will be POST-ed to your callback URL as a form-url-encoded content.

Keep in mind, that you have only 10 seconds, including the GpsNose server contacting your website and your response traveling back to GpsNose, as your user is in the middle of token-scanning, holding his mobile and watching the hour-glass ;-)!

When the user is offline because there is no internet connection, the scanned-in token is saved in his mobile and will be submitted as soon as possible – but this could be also hours or even days later. You see all the timing details in these fields:

- *createdTicks* – when was the token-definition created by you
- *scannedTicks* – when was the token scanned by the user
- *recordedTicks* – when GpsNose platform recorded the scanned token, i.e. the app was able to submit the scanned token back to the GpsNose server

When you need to map the received token to the original creating properties, like expiration, author etc., you must save the original token on your behalf and map it based on the payload you get from the scanned token.

There are 3 kinds of the callback params:

- *Token-definition*: these were defined by you when you've created the token
- *Scan-event*: these are given by the scan-event
- *User-supplied*: these can be optionally entered by the user, when the token's *options* allow it

NOTICE: We strongly recommend using the *https* version for your callback-handler, so the callback data is encrypted when POST-ed to your website. It's your responsibility to handle and store the token data securely, although the contained data should contain no private data.

3.7.4 Read-in all the tokens: async

You have 14 days to pick up all the tokens scanned-in by your users. The tokens are being recorded regardless of the sync/async scenario, so you can relax and work through them also if your callback was not working while your user scanned a token.

When the token's *CallbackResponseHttpCode* equals to 0, you know this token was never processed. If it's 200, it was processed synchronously while scanning (as you defined the mashup's *Callback URL* in the Mashup-Admin) and if it contains anything else, it was an error while sync processing.

The *CallbackResponseMessage* contains anything you've sent back from your callback (and your user saw this message), or it holds the error message, when the *CallbackResponseHttpCode* was set and it's not 200 (and your user saw just a general "Callback failed" message, not this specific technical error string).

```
var tokensApi = loginApi.GetMashupTokensApi();

var tokens = tokensApi.GetMashupTokensPage(_site.Community);
while (tokens.Count > 0)
    tokens = tokensApi.GetMashupTokensPage(_site.Community, tokens[tokens.Count -
1].ScannedTicks);
```

3.7.5 Batching

When the token's *options* have `BatchScanning` bit set, the mobile app allows scanning more QR-tokens and waits, until the user has completed the while token-batch. This allows shopping-cart like behavior: the user scans all the pizzas and then completes the order.

The whole token-batch is sent to the GpsNose platform in one step – but is sent item-per-item synchronously (when you've defined the callback-URL for you mashup) or asynchronously (when polling by your mashup site) back to your mashup site.

You can know a batch is coming, when some singular token-record has the `batchCreationTicks` set != 0. In such case, you must wait for more records to come (with the same `batchCreationTicks`), until the last record has set the `isBatchCompleted`.

Why is it designed like this?

- As otherwise, you would need to define 2 callback-URLs and then could come questions like: "what if a batch with single-item was scanned?" and "I don't need batching: why is it there?" and "do I need to define both URLs?" etc.
- As the batch could hold pretty much data and we don't know how responsive your server is: when breaking it down to item-level instead of the whole batch, the design with timing/error-handling/states etc. is easier for you to manage.

3.8 Administrative Tasks

3.8.1 Managing mashups

To create a new mashup community, you must:

1. Register a new mashup community, i.e. the desired bridging website
2. Verify the ownership of the registered mashup website

Later, you can update the mashup properties or re-generate the app-key, should it get disclosed to the public.

You can also list the mashups, which you have successfully created before.

```
public void Wiki_ManageMashups()
{
    var api = new GnApi();
    var adminLoginApi = api.GetLoginApiForAdmin();

    // login as admin for all your mashup sites
    var qrCode = adminLoginApi.GenerateQrCode();
    // TODO: scan the qrCode by your mobile-app
    adminLoginApi.WaitForLogin().Wait();

    // the admin API for managing the mashups
    var adminApi = adminLoginApi.GetAdminApi();

    // create a new web-community, i.e. "mashup"
    var validationKey = adminApi.RegisterCommunityWeb("%www.geohamster.com");
    // TODO: place the key into your root-dir or as meta "gpsnose-validation-key"
    adminApi.ValidateCommunityWeb("%www.geohamster.com");

    // when needed, change the app-key, e.g. when disclosed
    var newAppKey = adminApi.RegenerateAppKey("%www.geohamster.com");

    // you can always update the mashup properties when needed, e.g.
    // - allowing (or deleting) only localhost and some IP to connect
    // - setting (or deleting) a callback URL for sync-notif on QR-token-scanned
    adminApi.UpdateCommunityWeb("%www.geohamster.com",
        new List<string> { "localhost", "11.22.33.44" },
```

```
        "https://www.geohamster.com/tools/callbackFromScanningQrCode");

    // it's nice to be able to list all own mashups like this
    // it contains all the mashup details like quotas, sub-sites, member-counts etc.
    var myMashups = adminApi.GetOwnMashups();

    // create a private sub-community "secret" for the GeoHamster mashup
    // allow members to use the forum, invite others, but not to list other members
    adminApi.AddSubCommunity("*www.geohamster.com@secret",
        GnCommunityAcl.CommentsFromMembers | GnCommunityAcl.MembersInviteMembers);

    // now remove the sub-community, as it's not really needed ;-)
    adminApi.DelSubCommunity("*www.geohamster.com@secret-level");
}
```

3.8.2 Managing sub-communities

Once your mashup's main-community exists, you can add new sub-communities to it. Every such sub-community has its own ACLs, joined members, thumbnail etc., which can be managed directly from the mobile app.

You can also delete existing sub-communities, but only if the sub-community has *no joined members*. Just imagine: you have reached already the second level in GeoHamster, so you're a proud member of the "%www.geohamster.com@level-2" community. One nice day the community is deleted, without asking you – that would be a straight manipulation of your profile's communities.

For code sample, see the example in the previous chapter *Managing mashups*.

NOTICE: Should you *really* want to delete the sub-community having already joined members, enter the community editor in the mobile app and remove the members one by one.

3.9 Talks

A *Talk* is a sophisticated "area-chat" nearby, can have an existing topic selected or created a new one, can have a defined distance and time for the reachability control.

The *Talks* are currently not managed by the SDK to avoid automated SPAM message spreading. We are looking into this and are open to your feedback.

4 Appendix

4.1 Links

For the end-users

- Home-page: http://www.gpsnose.com
- User-guide: http://www.gpsnose.com/home/doc
- iPhone app: https://itunes.apple.com/us/app/gpsnose/id892215768
- Android app: https://goo.gl/4q4TGI

For the developers

- Wiki: http://wiki.gpsnose.com

- Mashup-admin: http://www.gpsnose.com/Developer/MashupAdmin
- Download SDK/CMS: https://wiki.gpsnose.com/download/

4.2 Versioning

Version	Changes	Date
0.1	ADD: Initial release	2018-11-11
0.2	ADD: Mails read/send ADD: Administrative tasks	2018-11-18
0.3	ADD: MashupToken extensions	2019-04-07
0.4	ADD: MashupToken extensions	2019-05-12