NobleHour API Quickstart

Introduction

Our API (Application Programming Interface) allows users to customize and tailor their own features using the NobleHour platform. It provides you with a framework to develop your own tools, specifically geared to the needs of your organization. Our growing library of routines integrate various NobleHour tasks, interactions, or functions with your organization. We provide the tools and best practices necessary so that NobleHour functions on any device, operating system, or application. All that we ask is that you provide the imagination!

What can I do with the NobleHour API?

Having worked with organizations such as schools, businesses, and non-profit entities, we continuously encounter situations where certain software functions require custom development for an organization's operational needs. Using our **API Endpoints,** easily create and manage NobleHour information. We've also included a search component that allows keyword, location, and relational queries. Our API is designed with REST principles. Later in this document, we'll also discuss our SDK (Software Development Kit) that works on top of REST API endpoints.

The NobleHour API is our development ecosystem. As we continue to update our libraries, we encourage suggestions from our users on how we can help you build applications that benefit your organization.

Our API enables interoperability between an organization's system(s) and NobleHour. To get a better idea as to how to envision this, let's consider the following scenarios:

> *While on campus, a student is searching for community service opportunities for extracurricular credit. Using their iPhone, they quickly access the online calendar system for the latest updates around campus, including NobleHour events and opportunities.*
>
> *A registered organization using NobleHour recently started a campaign to recruit volunteers for an organized food drive. Volunteers register through the store's webpage, which immediately retrieves the latest event times, dates, and locations from NobleHour.com.*
>
> *A non-profit organization taking part in a mobile blood drive must schedule and track the hours of volunteers working shifts throughout the month. Using their own organization's custom scheduling application for the iPad, hours can be recorded. This data is also synced directly to the organization's NobleHour.com account.*

Before Getting Started

To use our API, you'll need to be familiar with following:

> **HTTP** (Hypertext Transfer Protocol)
>
> **REST** (Representational State Transfer). To learn more about REST principles, checkout the REST API Tutorial.

Understanding the NobleHour API

1. Get acquainted with NobleHour's API by reviewing the **NobleHour API Basics** section. This provides an overview and glossary on how we define and associate objects when working with the API.

2. Review the next section, **NobleHour Endpoints**. Each API endpoint represents a set of resources within NobleHour. We'll breakdown the capabilities of each, giving you an idea of the scope of custom resources available to you.

3. In **Accessing the NobleHour API**, we'll walk you through the initial development stages and help you get started.

4. Be sure to check out our software development kits in the final section, **Our SDK (Software Development Kit)**.

NobleHour API Basics

Our database uses a "graph" approach to model relationships between multiple things and is our primary way of retrieving data. It's both HTTP-based and domain specific, allowing you to perform tasks such as query data, log hours, schedule events, or create opportunities.

When envisioning graph theory, **vertices** and **edges** are used to distinguish and associate these relationships. In the NobleHour database, we use different vertex types and different edge types to model these relationships.

> **Vertex**: A generic node in the graph (can represent one of many things).
>
> **Edge**: A connection between two vertices.

Vertices and edges provide a relational way to model and link our API endpoints together. You can create vertices to associate information such as organizations, groups, and projects. This is what we can an **entity**. Users then submit their information to this entity, otherwise known as **content**. Content represents various information that would be inputted on the user's side. For example, logging hours or locations into NobleHour.

> **Entity**: A vertex that is formed by an administrator in order to relate users and content. Specifically: Organizations, Groups, and Projects.

> **Content**: Vertices created by users for submission to an Entity. Specifically: News, Events, Opportunities, Media, Hours, Donations, and Goods & Services.

For each vertex type or edge type we have assigned a **Universally Unique Identifier (UUID)** and description. When you request for a vertex or edge, you'll include this ID. UUID is defined by RFC 4122,

ISO/IEC 9834-8:2005, and related standards. In some systems, this term is UUID is referred to GUID (Globally Unique Identifier).

Example

POST / organizations/ :id/groups

Where the **:id** would be replaced by the UUID of the organization.

An edge type of 0 (Parent) would be created between the group and organization.

NobleHour Vertex Types

Type	ID	Description
Address	11	
Asset	12	
Customer	6	
Event	5	
Group	1	A subset of users within an organization
Hours	10	
News	0	Content like a blog post or a youtube video
Opportunity	3	
Organization	2	
Organization (Offline)	4	
Submission	8	A request to moderate something
Submission History	9	
URL	13	
User	7	An account representing a person using NobleHour

NobleHour Edge Types

Type	ID	Description
Parent	0	Parent in an Organization/Group/User Hierarchy
Content	1	Content that has been approved/accepted (via Submission process)
Admin	2	An approved admin user role for a given vertex
Moderator	3	An approved moderator role for a given vertex
Contributor	4	An approved contributor role for a given vertex
Citizen	5	An approved citizen user role for a given vertex
Follower	6	Denotes the follower relationship for a given vertex
Like	7	Facebook/Google+ style such as “like” or “+1” or “star”
Author	8	Indicates which User has created the other vertex
Verifier	9	Indicates which User has verified the given Submission
Submission	10	Indicates the vertex that has received a submission
Hours	11	Set of hour log entries (for a Submission)
Change	12	Log of changes made to an Hours Submission (ordered by time)
Location	13	Indicates a location for the given vertex

Additional Terminology

Term	Description
Content	Stuff that is created & contributed by users, e.g. “News”, “Opportunity”
Destination	Refers to the entity that received a submission
Role	Establishes the permissions of a user with a vertex.

Accessing the NobleHour API in 3 Steps:

Note: For mobile clients, application code will need to utilize the API with the appropriate credentials.

Step 1.

Point the client to the appropriate environment base URL from the list below. Typically, the "Staging Environment" is commonly used. Depending on your needs, use an integration or a personal development server to point your client to.

Environment	URL
Development	Varies
Integration	https://integration-api.noblehour.com
Staging	https://staging-api.noblehour.com
Production (1.x)	https://api.noblehour.com
Production (2.0 temp)	https://developer.noblehour.com

Step 2.

In order to access private endpoints and obtain authorization tokens, please configure your client credentials. These will be provided by our support team upon activation.

Step 3.

Obtain tokens for user-scoped access. Use the **Authentication** endpoint to create, refresh, or revoke a token.

Obtaining Client ID Authorization

Authorization for the NobleHour API is done using **OAuth 2.0**. OAuth is a protocol that lets external applications request authorization to private details without storing passwords. This is preferred over Basic Authentication because tokens can be limited to specific types of data, and can be revoked by users at any time.

Clients by default have some access to publicly available information - similar to anyone browsing the NobleHour.com website itself. To obtain access to information on a user's behalf, an application must request the user's permission using the OAuth 2.0 protocol. A new authorization token is provided and may be used to gain further information as authorized by the user. A valid authorization token is needed for each request requiring access to secure, non-public data. Once obtained, the token may be used repeatedly until it expires.

What is OAuth 2.0 Protocol?
OAuth is a process for owners to authorize 3rd party access to their server without the need to share their credentials. On behalf of the resource owner, OAuth allows access tokens to be issued by third-party clients with their approval. For example, each day users log into 3rd party websites such as

Facebook or Twitter without the need to worry about security credentials thanks to the open standards that OAuth provides.

To learn more about OAuth 2.0 RFC please visit the resource page here.

Grant Types
Currently we only support two grant_types. These are **password** and **refresh**. Additional types are in development and being added as necessary.

Obtaining Tokens
To obtain an authorization token, a grant_type, username, password, and client_id must be encoded in JSON and sent via POST to the following url: https://api.noblehour.com/oauth/token

Attribute	Range	Notes
grant_type	password, refresh	Always password for first token request
client_id	a-zA-Z0-9	Provided by NobleHour
Username	any valid email address	
Password	user password	

Example body:

```
{
"grant_type":"password",
"username":"batman@gotham.com",
"password":"jokersmells",
"client_id":"aB23rsdfkj11ff"
}
```

A successful token request will result in a JSON response body that has access_token, token_type, expires_in, and refresh_token as attributes.

Example JSON response:

```
{
"access_token":"1245Fadsf",
"token_type":"Bearer",
"expires_in":3600,
"refresh_token":"asv2qrdsfR"
}
```

The access_token attribute may be used to access the API by supplying it as the Authorization HTTP header value.

The token_type should always be Bearer, meaning it acts very much like a simple random cookie token for sessions.

The expires_in value determines in seconds how long the token is valid.

Refreshing Tokens
The refresh_token may be used to obtain a new token and invalidate the old token at any time.

A refresh token request requires a POST body that looks like the following:

```
{
"grant_type":"refresh_token"
"refresh_token":"asv2qrdsfR"
"client_id":"aB23rsdfkj11ff"
}
```

The response will look the same as a password request, e.g.:

```
{
"access_token":"1246Gadsg",
"token_type":"Bearer",
"expires_in":3600,
"refresh_token":"btu3qrdsfS"
}
```

Authorized Requests

Once a client has authorized, requests should include the Authorization header in subsequent requests. For more information on header format, refer to the RFC on HTTP Authentication: RFC-2617.

Website Flow
Clients authorized directly via the API can transition a user to an authenticated session on NobleHour.com by posting to **/login** with a valid access token in the Authorization header. The server will validate the token and redirect the now logged-in user to the appropriate destination on NobleHour.com.

You're Done!

Now that you have access to our API, you can perform the following:

Use your tokens with an SDK configuration

Our SDKs are tools that allow organizations a flexible way in creating applications for specific software and hardware platforms. The SDK works directly with the NobleHour REST API, using library routines to call API endpoints.

Our SDKs support the following languages:

- JavaScript: https://github.com/treetopllc/noble.js
- Golang: https://github.com/treetopllc/noble-go-sdk
- Objective-C: https://github.com/treetopllc/noblehour-objc-sdk
- Java: https://github.com/treetopllc/noble-java-sdk

OR

Access our API Endpoints

These are organized by various categories, each with a specific list of functions to enable within your application (as permissions allow). You can access our Endpoints at http://treetopllc.github.io/redox/.

Revision History

6.11.2015	Document approved for circulation.