Andy Gu
Mail: andygu@usc.edu
IRC: everykittysdaydream

Title Open Supporter Data Integration

Synopsis

The Open Supporter Data Interface (OSDI) is a set of standards which aims to define a common interface for sharing data in a common format between progressive and nonprofit organizations. Currently, there is no reliable CiviCRM extension that allows organizations to implement the OSDI API across the platform. This project will create an extension that allows organizations to easily import data from external endpoints which are compliant with the the OSDI standard. This permits users to sync contact data across different platforms, setting CiviCRM as a single "source of truth" against other services such as Action Network, ActBlue, etc.

Organizations would be able to supply the URL of a third party service and a key. Through this extension, users that download this extension would be able to easily import contacts, people, events across different sources. Additionally, this project consists of an OSDI library that empowers developers to easily access OSDI-compliant datasource and read existing database data in OSDI-compliant JSON format.

CiviCRM has multiple import functions in extensions and core functions that allow users to import csvs of volunteers, people, events, contacts, etc. This project could also add options to import data through the provided extension.

Benefits to community

There are over ten thousand nonprofit or governmental organizations that rely on CiviCRM, managing more than 140 million contacts (https://civicrm.org/blog/cividesk/how-many-organizations-use-civicrm-where-how). Half of these organizations are in the United States.

Integrating OSDI support would empower CiviCRM users to easily manage data across multiple organizing platforms. For developers, OSDI integration cuts down on time spent building tools translating between heterogeneous services. For end users, data-entry personnel can save time otherwise spent doing manually data entry in OSDI. Additionally, compatibility makes CiviCRM a more attractive vendor because of potentially easy integration with third party services. The extra functionality makes CiviCRM interoperable with other tools for running electoral and community organizing campaigns, and a more attractive organizing tool itself.

Deliverables

Our deliverable includes:

Abstract data structures that represent all OSDI data-structures, and a library of functions that call out to these data structures via the CiviCRM API.

Navigating to the OSDI endpoint would open a drop-down menu in PHP where users would supply from a list of OSDI-compliant API endpoints and input the relevant data in a list of parameters.

Additional “import from OSDI” tools on at least the following options that CiviCRM currently provides:

- Import Contacts
- Import Activities
- Import Participants
- Import Memberships
- Batch Data Entry (Contributions)

These are the following OSDI analogues for these objects:
Import Contacts requires the remote server to support accessing the People resource, and preferably access to groups of People as a List object.
Import Activities requires the remote server to support the Events resource.
Import Participants requires the remote server to support the People resource.
Import Memberships requires the remote server to support the People resource.
Batch Data Entry requires the remote server to support the Donations and Fundraising Pages resource.

The profile for each activity is formatted along the descriptions here:
http://opensupporter.github.io/osdi-docs/list_exchange_profile.html.

In order of priority, it is most important to support accessing the People resource.

Architecture

This project is built as a CiviCRM Extension. Currently, our architecture consists of an **Abstract Importer class**. Future implementations, like an ActueBlueImporter or an ActionNetworkImporter will inherit from this class The Abstract Importer class contains the following functions:

```
abstract public function pull_endpoint_data();
abstract public function update_endpoint_data($date);
abstract public function validate_endpoint_data($data);
abstract public function add_task_with_page($page);
```

When pull_endpoint_data or update_endpoint_data are called, the **inherited Importer class** makes a network request to a third party API endpoint that is specified in a config.php file. This endpoint will be configurable by a user on another page. Additionally, any provided API keys would be added by the user in the frontend and loaded by the config.php.

The second component consists of two **Queue** mechanisms. The queue is managed by a CRM/Queue/Runner class, a CRM/Queue/Helper class, and a CRM/Queue/Tasks class. The first Queue manages individual pages of resources that have been taken from the 3rd party API. The second Queue manages individual requests to be made to the 3rd party API. On each cron run, the second Queue will request a page of resources and insert them into the first Queue. The first Queue will take all the resources in itself and insert them into CiviCRM using the CiviCRM APIv3.

A CiviCRM page contains an instance of and manages calls to the importer class. The OSDI landing page will have an “Enable Sync” button. When clicked, the page will call pull_endpoint_data and then set up a cron job to call update_endpoint_data daily from the inherited importer class.

Our architecture will also consist of an **Abstract Exporter** class from which all future endpoint exporters will inherit. This class will potentially contain the following functions:

```
abstract public function push_endpoint_data();
abstract public function reveal_updated_data($date);
```

The push_endpoint_data function will return all users in CiviCRM in a hal/json format. The reveal_updated_data function will do the same thing, although it will apply a filter over contacts in our CiviCRM instance which were only modified after a certain date. The exporter merely provides an API for other users to retrieve information from CiviCRM. To access this, users must provide credentials as a user or admin of CiviCRM.

Schedule

Date	Task
4/23 - 5/14	Community Bonding, Ramp up further with the CiviCRM codebase and extension development. Select a HAL PHP Library for transforming OSDI data.
5/20	Basic Abstract Classes for the People, Events, and Donations (stretch goal)

	resource are built. Basic Abstract Design of the “Importer” Class is built. This class would provide basic functionality that supports the use cases of an OSDI Importer.
6/5	Have a working script that imports People resources from an Action Network Group into CiviCRM. This can be a PHP script. 1. Webform UI page for people to submit a request to ActionNetwork 2. Backend functions that pull People resources from Action Network 3. Validator functions with a library (i.e. Python voluptuous) to make sure data isn't corrupt 4. Function that pulls every user from Action Network into CiviCRM 5. Functions that pull every user that has been changed given a certain date from Action Network into CiviCRM 6. Cron jobs set up to update changed or new users in Action Network 7. Dedup rules written to adapt to this page. PHPUnit tests written for every script developed.
6/14	Have a working script that exports new contacts added in CiviCRM into Action Network. This can be a PHP script. Open API endpoint that allows third parties to request all data from our local CiviCRM instance. Supported queries to request all CiviCRM data after a certain date. Provide a frontend display that calls out to the script. PHPUnit tests written for every script developed.
6/22	Have a working script that imports Actions from the Action Network Events API into

	CiviCRM. Provide a frontend display that allows end users to load provided actions PHPUnit tests written for every script developed.
7/1	Support for import functions as a job to be configured in cron.php is supported.
7/11	Have a working script that exports new events in CiviCRM as actions in Action Network.
7/22	Generalize the API for different endpoints. Provide support for import / export of Donations resources from the ActBlue Donations webhook into CiviCRM.
[rest]	Buffer Time

Profiles - Accessing a List of People

In this scenario, CiviCRM needs to import a list of people from another OSDI compliant application. CiviCRM takes this data and stores it in its own instance.

The steps in this profile are:

1. The user supplies an endpoint of choice and a corresponding APIKEY
2. The CiviCRM instance calls out to the endpoint to query the list of lists that endpoint provides.
3. The user selects a list displayed from the endpoint response.
4. The CiviCRM instance requests the list selected.
5. For each contact, the CiviCRM instance requests the People object and imports all new resources into its own instance.
    a. Unique resources will be identified and use the oData filter query to label unique resources if needed. Dedupe rules will be configured for new Person resources in CiviCRM.

All API exchanges are described with a Github gist here.

Profiles - Accessing a List of Events

In this scenario, CiviCRM needs to import a list of events from another OSDI-compliant application. CiviCRM takes this data and stores it in its own instance.

The steps in this profile are:

1. The user supplies an endpoint of choice and a corresponding APIKEY
2. The CiviCRM instance calls out to the events endpoint to query the list of events that endpoint provides.
3. The user selects a list displayed from the endpoint response.
4. The CiviCRM instance requests the list selected from the endpoint.
5. For each [contact?] member, the CiviCRM instance requests the Events object and imports it into its own instance.
    a. Unique resources will be identified and use the oData filter query to label unique resources if needed. Dedupe rules will be configured for new Event resources in CiviCRM.

All API exchanges are described with a Github gist here.

Profiles - Accessing a List of Donations

In this scenario, CiviCRM needs to import a list of donations from another OSDI compliant application. CiviCRM takes this data and stores it in its own instance.

The steps in this profile are:

1. The user supplies an endpoint of choice and a corresponding APIKEY
2. The CiviCRM instance calls out to the endpoint to query the list of fundraising pages that endpoint provides.
3. The user selects a fundraising page displayed from the endpoint response.
4. The CiviCRM instance requests the fundraising page for the collection of donation resources selected from the endpoint.
5. For the selected page, The CiviCRM instance requests the Donations object and imports it into its own instance.
    a. Unique resources will be identified and use the oData filter query to label unique resources if needed. Dedupe rules will be configured for new Donation resources in CiviCRM.

All API exchanges are described with a Github gist here. We need the following endpoints to be live on a given external endpoint for our functions to work.

Related Work

A previous GSoC student, Anudit Verma, has created a similar project here: https://github.com/anuditverma/org.civicrm.osdi.

However, there are a few important differences:

1. It currently acts as a server that inputs / outputs a new record to the system. But it only allows people to supply People resources. OSDI has a litany of other resources that could support campaign, event, and fundraising APIs and data structures that are OSDI compliant.
2. It is also fairly technical to set up, especially from the perspective of an end user. For instance, supplying an API_KEY requires either a second extension or some basic knowledge of PHP.
3. It does not provide immediate support for importing data from external OSDI compliant endpoints.