DEIP Technical Report (Phase 2)

Angus McLeod, October 2022

Introduction **1**

DEIP Plugin **2**

Features and Documentation 2

Providers 2

Sources 3

Events 3

Connections 3

Logs 3

Tests, Linting and CI 4

OmniEvent **4**

Provider Gems 5

omnievent-icalendar (RubyGems) 5

omnievent-api (RubyGems) 5

omnievent-eventbrite (RubyGems) 5

omnievent-eventzilla (RubyGems) 5

omnievent-meetup (RubyGems) 5

Tests, Linting and CI 5

Introduction

This report is a summary of the technical work conducted in Phase 2 of the Discourse Events Integration Plugin (DEIP) project in Next Generation Internet's (NGI) Data Portability and Services Incubator (DAPSI) program. This report assumes familiarity with the DEIP Research Report submitted at the end of Phase 1 of the DEIP Project.

The goals of Phase 2, as specified in Pavilion's DAPSI Workplan were

3. Developing the prototype into an MVP (4.5 months)
We will develop the specification prototype into an MVP that integrates with existing Discourse features and plugins to allow it to be used for a variety of real-world use cases. We will add CalDAV support to allow for easy event export.

Deliverable: A working DEIP MVP

4. Packaging, launching and promoting the Product (1.5 months)

We will package the plugin as a product, add a business-level subscription, publish landing pages and documentation, and promote the product both amongst the Discourse community and to communities not yet using an open source solution for event management.

The work unfolded broadly as predicted and the substance of the deliverables were achieved. The DEIP has been released and is currently being tested in a closed beta amongst Pavilion's existing customers and also in an open beta in the Discourse Community. The OmniEvent framework implementing the standard developed in Phase 1 is also "pre-released" on rubygems.org (it was "in development" at the end of Phase 1) following various improvements resulting from the work on the DEIP itself. Finally, Pavilion will likely be presenting our "Practical Event Model" standard at the CalConnect conference in Nottingham in November, as a first step to pursuing formal standardisation. All of the main actors in calendar event standardisation will be at CalConnect.

This report briefly will summarise the technical development and result of each of those elements. It is accompanied by a video containing an overview of the same. The demonstration forum shown in the video can be accessed at demo.pavilion.tech. An administrator account to that site can be provided on request.

DEIP Plugin

The development cycle of the DEIP plugin was mostly standard for Discourse plugins, with the slight difference being the high number of custom gem use due to the OmniEvent framework. The author is preparing a pull request to Discourse itself (discourse/discourse) to improve the experience of developing a plugin with multiple gem dependencies.

Features and Documentation

The DEIP features are what you might expect from a plugin that integrates event providers with a framework like Discourse. Event administration is accessible to administrators via the Discourse administration panel and broken into the sections described below.

These descriptions should be read along with the first version of the documentation for the DEIP which we've published for site administrators participating in the DEIP beta. That documentation is focused on functional use. We will focus on the technical aspects and overall narrative here. Note that the documentation is published on a Discourse plugin directory which Pavilion developed and is currently trailing in partnership with other actors in the Discourse ecosystem.

Providers

To get a basic understanding of "Providers", and a list of currently supported providers, please first read "Add a Provider" in the documentation. "Providers" is how an administrator connects to different event providers and handles any authentication that may be required to access their event data. Underpinning this concept is data collected in the "Providers" and "Authentication" worksheet of the DEIP Research Data prepared in Phase 1.

It's worth mentioning that we decided to not handle authentication within the OmniEvent framework to keep that as abstracted and focused as possible. There are various ways provider connection and authentication can be handled, the "Providers" functionality in the DEIP being one example (and it will be used as an example when explaining this aspect of the standard - see further below).

Sources

A provider can have event data in multiple places, may have different types of events (e.g. "event series") and may change both of these over time. This is why we made the decision to separate "Sources" of events from providers, a concept which roughly translates to the "Request" worksheets of the DEIP Research Data. Please see "Add a Source" to understand more about the functionality of Sources.

Events

Once you've got a provider and a source you can "import" events. The importation process relies on OmniEvent to retrieve and format the events. It stores the retrieved events in a separate database table to allow for flexibility of integration and administration. It also means an admin can get a handle of what events they'll be integrating into their community before they appear to their users. Please see "Event Management" in the documentation for more.

Connections

A "Connection" is where events imported from a source actually get added into the community and appear to users. Currently, Connections are based around Pavilion's existing experience with community event management, namely that events typically live in dedicated categories, e.g. a category called "Events". As such the connection an administrator can make in the first version of the plugin is to "Sync" the events they imported into a designated category. The admin can specify the salient aspects of how they might want the event to appear, in particular the poster and the event "Client".

With respect to the event "Client", in the current version of the DEIP, the available clients are the existing event-related plugins for Discourse: Pavilion's own "Events Plugin" and Discourse.org's "Discourse Event" plugin. This means that the DEIP automatically works with whichever client the site administrator is currently using to administer events in their community.

Our plan is to assimilate Pavilion's existing Events Plugin into the DEIP and brand the combined plugin as the "Events Plugin" for Discourse. We haven't yet taken the step of assimilation as we felt that would have been an engineering step too far prior to testing the DEIP functionality in our closed beta. The combined plugin will continue to support the "Discourse Event" plugin as a client for maximum interoperability.

Logs

The DEIP has its own dedicated logging system in addition to the standard Discourse and Rails logging systems. This dedicated system is necessary for two reasons

- It surfaces specific and descriptive provider integration, event import and connection sync errors to the site administrator
- It integrates with OmniEvent's logger, allowing for specific and descriptive OmniEvent-level errors to surface to the site administrator.

When integrating multiple separate providers, there are quite a few potential sources of error and we considered it important, both from a technical perspective and a UX perspective to make the logging system descriptive and specific.

Tests, Linting and CI

The DEIP is covered in unit, integration and acceptance tests, following the best practices in Discourse plugin development, specifically rspec for the backend (Ruby on Rails) and QUnit for the frontend (Ember.js). The DEIP also follows the best practices in Discourse plugin development code standards including:

- Strict use of Discourse server-side and client-side versioned APIs to ensure long-term stability.
- Use of eslint, prettier and rubocop to format the code, using the same configurations used in Discourse itself.

Also, the plugin utilises github workflows for continuous integration (CI). All tests, formatting checks and version checks are run on:

- every pull request (feature development and major fixes)
- every direct commit to main (minor fixes); and
- on a cron schedule to ensure continued compatibility with Discourse itself.

The last CI flow is necessary because, unlike Wordpress, the default Discourse installation is not versioned in a traditional sense. It uses the "test-passed" branch of the "discourse/discourse" repository and site administrators get prompted to update their installation as soon as they are behind the latest commit to that branch. This means there is a higher-than-normal likelihood of incompatibility issues arising for a plugin in the Discourse ecosystem. It is why high code standards, an appropriate test suite and CI is critical to the viability of a production level plugin in the Discourse ecosystem.

OmniEvent

The development of OmniEvent (RubyGems) continued alongside the development of the DEIP, with various improvements and functionality tweaks being made as OmniEvent was put to use in the real use case provided by the DEIP. The OmniEvent gem and its various provider-specific gems are all currently in "pre release" which is the standard for Ruby Gems ready for beta (e.g. "omnievent" is currently 0.1.0.pre3), but not yet ready for production.

Provider Gems

For context, it’s useful to recall that we are following the pattern of the “OmniAuth” gem which provides a general authentication framework in the Omniauth gem itself, and then allows providers to be integrated as needed via an ecosystem of provider strategy gems. In our Phase 1 Research Report we anticipated the development of three provider gems for OmniEvent in Phase 2 of DAPSI. We ended up developing five, and have a sixth in development, each being driven by requirements of participants in the DEIP beta.

omnievent-icalendar (RubyGems)

This connects iCalendar endpoints to OmniEvent. It’s worth mentioning that it supports iCalendar RRULE (recurring events), which the DEIP uses to support recurring calendar events.

omnievent-api (RubyGems)

This is a generic OmniEvent strategy to connect an API-based provider to OmniEvent. It will only be used as a dependency for provider-specific gems, abstracting common functionality needed when interacting with an events API.

omnievent-eventbrite (RubyGems)

This connects OmniEvent to Eventbrite. This is perhaps the most straightforward of the provider-specific gems, utilising the omnievent-api gem and adding eventbrite-specific requests and formatting.

omnievent-eventzilla (RubyGems)

Similar to the Eventbrite gem, this is a straightforward connection of EventZilla into the OmniEvent system.

omnievent-meetup (RubyGems)

This integration was slightly more complex due to the use of GraphQL in the Meetup.com API and the relatively scant Meetup.com Documentation on the same.

Tests, Linting and CI

All of the OmniEvent gems follow best practices for tests, linting and CI for ruby gems. This means:

- Using Rubocop for formatting
- Covering the gem in Rspec tests
- Using github workflows to run the linting and tests on every commit