# The rough guide to deploying shindig

## Setting up an OpenSocial container with Shindig

The following is a basic overview on how to integrate shindig into an existing social network site, to allow it to host OpenSocial applications.

This is currently just a rough draft, more like the table of contents of a guide than a guide itself.

We assume you already have a social network, that's to say, a site with registered users and a notion of "friends". Your site needs to be able to render customized pages for each user. Useful, but not required, is a way to send messages between users and a notion of an "activity stream" that lists actions that the friends of a given user have taken recently.

Sindig provides, on top of this infrastructure, a rendering engine for gadgets, a translation layer from your data to the OpenSocial data types, and an implementation of the communication protocols required for the gadgets to access that data.

Shindig does not provide an application directory, nor the specific pages inside which the gadgets will render.

## Getting shindig

You can

- Get the released version and dependencies. As soon as there is a release, of course.
- Or, you can get it from the source repository
  - download the 1.0 branch or tag
  - build with maven

## Interfacing shindig to your data

### The *Service interfaces

they need to

- Decode arguments (translate ids, filter names, any other changes they need from the OpenSocial formats to yours.)
- Perform validation & authorization that the backend does not handle.
- Send queries to the backend - translate the responses to the model.* interfaces.
- Filter & sort, if the backend didn't.

See Dave's architecture overview and Harry's instructions on providing your own shindig services for more details.

### Implement OAuth handling

See Enabling Oauth Support

### Guice Module

Write a Guice Module wiring all your implementations toghether.

### Configuration.

- config/container.js - Declare your list of supported fields for each datatype, configure url templates for proxying, navigation, etc. Configure use of secure tokens.
- common/conf/shindig.properties - http caching and rewriting config.
- Whatever configuration your classes need.
- Enabling Shindig Administration -> Shindig Administration

### Deployment

Build a WAR file. Take the one build in the server/pom.xml file of the distribution as an example. Deploy this WAR on an application server running in a different domain from your container. This is important, for security.

### Integrate with your site

- Create an application directory. This will list available applications, allow users to install/remove, track any sort of permissions for the application (Can it use the home view? can it send messages?, etc) and store the information from each app (url, oauth secrets, other metadata).

- Create a userPrefs mechanism.

- Create the container pages for the different views (home, profile, canvas) What views make sense depend on your site. The container page must generate an iframe for each embedded gadget, put whatever decoration your site requires around it, and a way for users to edit gadget preferences.

- Generate tokens for the iframes. (Code examples for this are needed).
- Install rpc_relay.html on your site to allow communication between gadget and container javascript. For example if you have shindig running on gadgets.example.com and your container is on www.example.com then you may copy it to http://www.example.com/gadgets/files/container/rpc_relay.html. Then you need to set the path to the new location in your container config file (either config/container.js or your local copy), in this example to the value "/gadgets/files/container/rpc_relay.html". The host name is automatically set in gadgets.js so you probably won't have to explicitly set it (though it doesn't detect https so if your container is on a secure page you'll need to use gadgets.container.setParentUrl() to tell it).