Organizational Identity Source Plugins

See also: Writing Registry Plugins

> ⚠ The interface requirements for Organizational Identity Sources is considered *Experimental*, and may change across minor releases.

Plugin Requirements

1. The name of the Plugin should match the format `FooSource`.
2. The Plugin should implement a `FooSource` model and a corresponding controller (`FoosController`).
   a. This model should `belongTo OrgIdentitySource`.
   b. The controller should extend `SOISController`.
   c. When a new Org Identity Source is created, a skeletal row in the corresponding `foo_sources` table will be created. There is no `add` operation or view required. The skeletal row will point to the parent Org Identity Source.
   d. When an Org Identity Source is edited, the entry point to the Plugin will be `foo_source/foo_sources/edit/#`. This will be called immediately after the parent Org Identity Source is created.
   e. Note `OrgIdentitySource` has a `hasOne` (ie: 1 to 1) relationship with `FooSource`.
   f. The table `foo_sources` should include a foreign key to `org_identity_sources:id`.
      i. Other tables used by the plugin should reference `foo_source:id`.
3. The Plugin should also implement a model named `FooSourceBackend`.
   a. The backend is responsible for the implementation of the backend search and retrieval capabilities.
      i. The raw record returned by the `retrieve()` function should not change if the underlying backend record has not changed. Registry uses the raw record to determine when the related Org Identity record must be updated.
      ii. The Plugin should support `mail` (email address) as a searchable attribute. This capability is used by Enrollment Flows in various configurations, see *Integration With Enrollment Flows*, below.
      iii. As of Registry v3.1.0, Plugins may also implement `getChangeList()`, which determines the set of changed records for a given time period. When supported, OIS sync processes in Update Mode will run more efficiently.
         1. As of Registry v3.2.0, if the `forcesyncorgsources` task is used, `getChangeList()` cannot be used.
      iv. See also *Supported Attributes*, below.
   b. This model should extend `OrgIdentitySourceBackend`, and implement the abstract functions defined in the parent model (see `app/Model/OrgIdentitySourceBackend.php`).
   c. Note that the Plugin configuration for `FooSource` will be available to the backend in `$this->pluginCfg`.
4. Registry will automatically track the current backend data via the org_identity_source_records table.

Supported Attributes

The Org Identity Source Backend is expected to return both a raw record (directly representing the backend datasource), and a formatted record. The formatted record is expected to represent an OrgIdentity, in typical Cake array format, along with its associated Models. The Backend may return the following supported attributes:

Attribute	Multi-valued?	Required?	Notes
Address	Yes	No	See note ℹ below.
EmailAddress	Yes	No	See note ℹ below.
Identifier	Yes	No	Does not automatically include the unique key (SORID). See note ℹ below.
Name	Yes	No	At least one Name must be returned, and exactly one Name must be flagged primary. See note ℹ below.
OrgIdentity.affiliation	No	No	Possible values may vary by CO; see `CoExtendedType::definedTypes`
OrgIdentity.title	No	No	
OrgIdentity.o	No	No	
OrgIdentity.ou	No	No	
OrgIdentity.valid_from	No	No	Must be in UTC time, and database format, eg via `strftime("%F %T", ...)`
OrgIdentity.valid_through	No	No	Must be in UTC time, and database format, eg via `strftime("%F %T", ...)`
TelephoneNumber	Yes	No	See note ℹ below.

Url	Yes	No	See note below. Available from Registry v3.1.0.

For multi-valued attributes, only one attribute of a given type is currently supported. For example, there can only be one `official` EmailAddress, though a second EmailAddress may be provided if (eg) of type `personal`.

Possible types may vary by CO, see `CoExtendedType::definedTypes` for valid types.

As of Registry v3.3.0, an identifier of type `SORID` will automatically be added to the OrgIdentity generated by an Organizational Identity Source Plugin, using the using the Plugin's record key (the same value passed to `retrieve()`).

Example

```
$myData = array(
  'OrgIdentity' => array(
    'title' => 'Researcher',
    'o' => 'University of Impossible Equations',
    'ou' => 'Department of Timey Wimey Stuff'
  ),
  'Name' => array(
    array(
      'given' => 'Pat',
      'family' => 'Lee',
      'type' => 'official',
      'primary_name' => true
    )
  ),
  // Note below here are multi-valued arrays
  'Identifier' => array(
    array(
      'identifier' => 'plee@university.edu',
      'type' => 'eppn',
      'login' => true
    )
  ),
  'EmailAddress' => array(
    array(
      'mail' => 'plee@university.edu',
      'type' => 'official',
      'verified' => true
    ),
    array(
      'mail' => 'plee@socialemail.com',
      'type' => 'personal',
      'verified' => false
    )
  )
);
```

Groupable Attributes

Is order to support Group Mappings for Organizational Identity Sources, the Plugin must implement two functions in the Backend file.

1. `groupableAttributes()` defines the set of attributes the Plugin knows about that may be used for generating group memberships. This may be a static list of attributes, or (as of v3.1.0) it may be dynamically determined based on a given instantiation (via the configuration available in `$this->pluginCfg`).
2. `resultToGroups()` converts a raw result into an array of attribute value/pairs. Note the array is not itself the group mapping, but rather the relevant attributes that will be used by the core code to determine if any group membership match. (This way the Plugin does not need to worry about parsing the mapping configuration.)

Prior to v3.2.0, the `resultToGroups()` was poorly specified, allowing either a single value or a list of values. Beginning with v3.2.0, the interface has been clarified, and the result is now an array where the key is the attribute and the value is a list of array, each of which may contain the following keys: `value` (required, holds the actual value), `valid_from`, and `valid_through`. If specified, the latter two must be in strftime %F %T format, and in UTC.

Note that while it is possible for a backend to return multiple entries for the same group with different validity dates, these must be consolidated down to a single CoGroupMember record. (While the data format theoretically allows multiple CoGroupMember records for the same CO Person in the same CO Group with different validity windows, in practice this is not supported anywhere.) If multiple entries are found, the group membership mapping code will attempt to pick the "best" one, which is generally the current record, or the one with the latest valid through date. Plugins can implement more deterministic algorithms by setting the results from `resultToGroups()` appropriately.

Example

```
public function groupableAttributes() {
  return array(
    'title' => _txt('fd.title');
  );
}

public function resultToGroups($raw) {
  // The core code will use this data to determine if the configured
  // OIS Group Mapping rules are matched.

  return array(
    'title' => array(
      array(
        'value' => 'Professor of Mysterious Events',
        'valid_from' => '2017-08-01 00:00:00',
        'valid_through' => '2018-07-31 23:59:59'
      );
    );
  );
}
```

Integration With Enrollment Flows

Organizational Source Plugins can be integrated with Enrollment Flows by way of Enrollment Sources. See the Enrollment Sources documentation for an overview of the various modes and how they are used.

Authenticate Mode

Most modes are supported using the interfaces described in *Plugin Requirements*, above. However, *Authenticate* mode requires OIS Plugins to implement an additional interface. (Plugins should use this interface to support *Authenticate* mode, and not the general Enrollment Flow Plugin interface, since this interface will automatically handle configuration checking and plugin ordering.)

The Enrollment Flow will hand off control during the `selectOrgIdentity` step to the entry point `foo_source/foo_source_co_petitions/selectOrgIdentityAuthenticate/#/oisid:##` (where # is the relevant CO Petition ID, and ## is the Org Identity Source ID).

The easiest way to implement this is for the Plugin itself is to extend `CoPetitionsController`. This way, most of the overhead of processing the request will be handled for you, and your plugin need only implement the function `execute_plugin_selectOrgIdentityAuthenticate`, where control will be passed. Once your plugin is finished, it should return control to the flow by redirecting back to the main flow, using the URL passed in `$onFinish`. The redirect URL is also available in the view variable `$vv_on_finish_url`.

Sample Source Plugin

```
// Plugin/FooSource/Controller/FooSourceCoPetitionsController.php

App::uses('CoPetitionsController', 'Controller');
class FooSourceCoPetitionsController extends CoPetitionsController {
  public $name = "FooSourceCoPetitions";
  public $uses = array("CoPetition",
                       // Your plugin will most likely need to use OrgIdentitySource to
                       // create the OrgIdentity
                       "OrgIdentitySource");

  /**
   * @param  Integer $id CO Petition ID
   * @param  Array $oiscfg Array of configuration data for this plugin
   * @param  Array $onFinish URL, in Cake format
   * @param  Integer $actorCoPersonId CO Person ID of actor
   */

  protected function execute_plugin_selectOrgIdentityAuthenticate($id, $oiscfg, $onFinish, $actorCoPersonId) {
    // Do some work here, then redirect when finished.
    // By default, Exceptions will be caught further up the stack, though you could catch them here.

    $myId = result_of_some_work();

    // Create an Org Identity
    $this->OrgIdentitySource->createOrgIdentity($oiscfg['OrgIdentitySource']['id'],
                                                $myId,
                                                $actorCoPersonId,
                                                $this->cur_co['Co']['id'],
                                                $actorCoPersonId);

    // Create some history
    $this->CoPetition->CoPetitionHistoryRecord->record($id,
                                                       $actorCoPersonId,
                                                       PetitionActionEnum::IdentityLinked,
                                                       _txt('pl.foosource.linked', array($myId)));

    $this->redirect($onFinish);
  }
}
```

Standard MVC rules apply.