Upgrades with Reference data

Problem Statement

When upgrading a FOLIO system by calling the Okapi tenant install API with a list of modules to upgrade, an operator may choose to specify the loadReference=true tenant parameter. This may be required, for example, for the tenant to take advantage of a new controlled vocabulary specified by a FOLIO SIG, or to get an update to an existing controlled vocabulary. As currently implemented in most modules, this will cause the module to attempt to load all reference data (not just new data). New records will be created if needed, and existing records (matched by UUID) will be overlaid.

This can lead to the following issues:

- If the tenant has altered or deleted any of the reference data loaded by the module when it was first enabled (which is possible in some cases both using the reference UI and using the module APIs), any changes will be overwritten with the system defaults, and deleted records will be re-created.
- If the reference data have data constraints (for example, the requirement that a particular property be unique), and the tenant has created a new reference record which causes a conflict with incoming reference data, the upgrade will fail, and the system will be left in an inconsistent state.

For more background, see: https://discuss.folio.org/t/reference-data-and-upgrades/2858; see also TC/Defining data types in FOLIO for automatic upgrades

Examples

- The reference data provided by mod-inventory-storage include labels in English. In order to provide labels in the local language, users must update the records (either in the UI or using the API). These labels are overwritten on upgrade.
- The reference data provided by mod-users includes patron groups. It is possible to manage those data in the users setting UI (and in fact it is perfectly reasonable to want to customize patron groups). Patron groups have a unique constraint that on the "group" property, so if the tenant has created a patron group that conflicts with incoming reference data, the upgrade will fail.
- The reference data provided by mod-inventory-storage includes a location hierarchy for Københavns Universitet. Locations can be managed in the tenant settings UI. It is very likely that a tenant will wish to remove the provided location hierarchy and create their own. Locations have unique constraints on the name and code. If the tenant creates a new location that conflicts with incoming data, the upgrade will fail.

Underlying issues:

We see this as a three-fold problem:

1. The upgrade fails if there's a problem with loadReference=true, and leaves the system in an inconsistent state. Should not fail, should report errors and continue.
   a. Note: Worse for non-RMB modules, ignore loadReference parameter and do their own thing
2. No way to get an output of what would be modified, the upgrade steps are all launched through Okapi fire-and-forget, so it fails or completes but no interactive component or ability to simulate.
3. There are no clear boundaries between the different types of data packaged with FOLIO, reference data vs. sample data. E.g. service points for DIKU tenant are loaded as reference data. A side effect is that tenants must edit the reference data according to local needs.
   a. This is a pitfall for upgrades, since modifications by users (which they are encouraged and enabled to through GUI) can be either breaking or being overwritten, so this is a potential data-loss scenario.

We do have UIs for editing Reference Data. Tenants are able, encouraged, and need to modify some of the reference data according to local circumstances. Upgrades should be able to preserve these changes while bringing in system-supplied updates.

Overall goal:

Upgrades have to be a fire-and-forget mechanism, without worrying about deleted custom data, outdated system data and so on.

The end result should not leave the system in an inconsistent or unusable state. Upgrades should continue unless failures encountered are truly fatal to the process and it cannot continue without a particular step succeeding.

Minor errors may happen and should be logged clearly for remediation by the system administrator.

Currently FOLIO delivers Reference Data and Sample Data, and these are not clearly distinguished. We would like to see:

1. A more rigorous definition of reference data.
2. The system should treat the base reference records as immutable, and introduce local updates as overlays on top of the base (like a customized view of the record for the tenant).

What follows is one possible model.

Proposal:

First, make a clear distinction between sample data and what we call 'Reference data' at the moment.

Second, split what we now call 'Reference data' into 3 layers or categories with different degrees of protection:
<table>
<thead>
<tr>
<th>Name</th>
<th>Usage</th>
<th>Overwritten on upgrade</th>
<th>Immutable (towards user)</th>
<th>Example of data stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>System data</td>
<td>Core information the module can not work without</td>
<td>YES</td>
<td>YES</td>
<td>PORT=9135; DEPENDENCY=mod_login, mod_permission</td>
</tr>
</tbody>
</table>
| Predefined Optional Working Set (POWS) | Predefined users/groups, schemas, $STUFF to work with | On admin request (LoadReferenceData=true) | YES | schema_book{
  • title(string,128)
  • subtitle(string,256)
  • authors(string,128)
  • publisher(string,128)
  • release_date(integer,4)
  • blurb(string,32768)
} |
| Custom data | Can override any given reference & system data, as well as introducing new values | NO (or only if the admin wishes to start with clean slate, like apt-get purge on Debian) | No | schema_book{
  • uuid(integer,64)
  • subtitle(NA)
  • authors(string,256)
  • release_date(string,64)
} |

--- Result ---
this is not a layer, just what the module works with in the end

| No layer | no layer | PORT=10587; DEPENDENCY=mod_login, mod_permission | schema_book{
  • uuid(integer,64)
  • title(string,128)
  • authors(string,256)
  • publisher(string,128)
  • release_date(string,64)
  • blurb(string,32768)
} |

User: JohnDoe (Admin)

System data is meant to consist only of information which causes the module to fail critically if not present. It may be altered by POWS or the custom overlay, but can never be not available, like a network port to use.

Predefined Optional Working Set (POWS) is a set of data that helps in using the module. Omitting it is possible, although basic functions of the module may not be available or working in an expected way (e.g. default user groups & permissions).

Custom data is a layer which keeps all users changes. It is also host to any example data which is loaded (like diku) and may not be altered during upgrades (notable exception: The admin explicitly wishes to overwrite EVERYTHING changed by his users, which will empty the entire layer). Even once introduced example data is not updated during upgrades to prevent any changes to custom settings.

This is introducing a transparent layered structure, in which the user will see data from all layers within the settings screen, and may alter everything there, although the changes will only be saved to the custom data layer. Modules should look for information in the following order, stopping at first hit:

1. Custom data layer
2. POWS layer
3. System data layer

Modules apply custom settings/data first, falling back to the POWS in case of missing information and further falling back to system data if necessary.

This way updating critical information and even POWS is possible without breaking the updating process, since both layers are untouched by users. Data which is stored in the custom layer may break some newly introduced features, making meaningful logging and error reporting towards the user necessary.