Arches4 Documentation

Release 4.0.0

Farallon Geographics, Inc.
Legion GIS, LLC

Oct 10, 2017

Introduction

CHAPTER 1

Overview

Documentation Overview

This is the general User Manual for Arches. It should provide you with background information on Arches, how to install it, and a good overview of its capabilities. While you are using Arches, be aware that much of the content here is also available by clicking the "?" symbol in the top-right corner of any page. If you are a developer interested in customizing or contributing to Arches, you should also visit our GitHub wiki for more information about how Arches works underneath.

What is Arches?

Arches is a web-based, geospatial information system for cultural heritage inventory and management. The platform is purpose-built for the international cultural heritage field, and it is designed to record all types of immovable heritage, including archaeological sites, buildings and other historic structures, landscapes, and heritage ensembles or districts.

Arches allows administrators to create their own database schema, and manage their own thesauri, while end users can search, explore and download the resources directly. In this way Arches is not only a robust and easy to use inventory system, it is also a perfect way to publish and disseminate your organization's cultural heritage information.

To try Arches out, you can view the v4 demo installation, go to the Arches project page for more background, or join our Google Group to engage with the active Arches community.

Arches is a web framework for Django and is designed to make it easier to build applications that need:

- **Geospatial data management** and geoprocessing like a GIS (Geograhic Information System) offers, but with a much more flexible approach for modeling the geometries associated with a resource.
- the ability to **import arbitrary data schema** in the form of graphs as a means of defining the set of attributes that describe data resources
- an **Ontology** as a means of formally naming and defining data types, properties, and the relationships between the data entities that describe a resource.

- **Thesauri** to manage the controlled vocabularies needed to describe and index information in a consistent and uniform way.

Arches manages data "resources". Resources can represent almost anything you want: physical things (such as a cultural heritage object), temporal things (such as activities or events), actors (such as a person or organization), or conceptual objects (such as an image. document, or other information carrier).

Resources are defined as directed graphs (nodes connected by edges). Nodes in the graph are used to represent the attributes (or collection of attributes) of a resource and edges define the type of relationship between attributes. In practice, a resource graph in Arches functions much like a schema does in a relational database.

Arches provides core services for creating, reading, updating, and deleting resources. Because resources are defined as graphs, Arches provides the services needed to import and parse resource graphs, as well the ability to create and interact with instance graphs (e.g.: an instance of a resource graph).

To promote consistent data creation, update, and indexing workflows, Arches implements a Reference Data Manager (RDM) that can manage thesauri. The RDM allows users with the appropriate privileges to update thesaurus entries in a manner compliant with SKOS (http://www.w3.org/2004/02/skos/) and assign the concepts within a thesaurus with data entry forms.

Arches' User and Developer forum: http://archesproject.org/forum/

Version History

v3.0 April, 2015: Updated architecture, inclusion of the Reference Data Manager, updated dependencies (ElasticSearch, OpenLayers, Knockout)

v2.0 March, 2014: Improved upload of digital files, assorted bug fixes

v1.0 October, 2013: Initial Release

License

Who is Arches for?

Arches is primarily intended for software developers who need to build flexible and responsive web applications, and who wish to hide the complexities of ontologies, thesauri, and geospatial data management from their users.

The Arches framework is largely invisible to end users; Arches applications are responsible for defining resource graphs and implementing data entry, data presentation, and data reporting workflows.

One major exception to this rule is the Reference Data Manager (RDM). The RDM is a core component of the Arches framework and can support the creation and curation of thesauri without the need to implement an Arches application.

Contributing To Arches

Arches is open source software, which means that with your help it will continue to evolve and improve. Contributions to the project can take a few different forms:

- **Translations** We are always hoping to bring Arches to new audiences around the world. Please get in touch from the main Arches website to express your interest in helping with translations.
- **Bug Reports** Reporting bugs is strongly encouraged, and can be made at the official Arches GitHub repository. Please read this article before reporting issues.

- **Code Contributions** For developers, good guidelines on how to contribute to the Arches code-base can be found in our GitHub repo wiki.

CHAPTER 2

Installation

System Requirements

Arches works on Linux, Windows, or macOS, but some of its dependencies may be more difficult to install on certain operating systems. Most enterprise-level installations of Arches have been created on Linux servers.

To begin development or make a test installation of Arches, you will need the following:

- **2gb disk space**
  - 1.5gb for all dependencies and 600mb for Arches.
  - In production, the amount of disk space you need will depend on the number of resources in your database, specifically uploaded images or media files.
- **4gb memory (RAM)**
  - This recommendation is calculated by the fact that ElasticSearch requires 2gb to run, and as per ElasticSearch documentation no more than half of your system's memory should be dedicated to ElasticSearch.
  - In development, it's possible to force ElasticSearch to run with only 1gb of memory, as we have noted here.
  - In production, you may want to increase your memory, and allow ElasticSearch to use more than its default.

Please note that Arches 4.0.0 release is only fully supported in Chrome.

Dependencies

- **PostgreSQL 9.6 with PostGIS 2.3** The easiest way to install this on OS X is Postgres.app. For Windows, use the EnterpriseDB installers, and use Stack Builder (included) to get PostGIS.

- **GDAL** and **GEOS** GDAL Version > 1.11.5. On Windows, use the OSgeo4W installer, and choose to install the GDAL package.
- **Python 2.7.x** Installation instructions: https://www.python.org/downloads/. We recommend 2.7.12, and on Windows you must choose 32-bit or 64-bit Python depending on your operating system architecture.
- **pip** If you are using Python < 2.7.9, you will need to get pip from here: https://pip.pypa.io/en/latest/installing.html. In Python 2.7.9 and later, pip is included by default. For good measure, you may as well upgrade pip right away with this command `python -m pip install --upgrade pip`.
- **Bower** Requires NodeJS/npm. Installation instructions here: https://bower.io/#install-bower
- **Mapnik** Version 2.2. Installation instructions: http://mapnik.org/pages/downloads.html. If you are on 64-bit Windows you will not be able to install Mapnik; the Python bindings are not compatible. Please read this note on our Wiki about how this limitation affects Arches.
- **JDK** Use these installers: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html. On Windows, be sure to set the JAVA_HOME system environment variable.
- **Elasticsearch 5.2.1** Technically speaking, Elasticsearch is a dependency. However, Arches comes with an easy way to install it, which is described in Installing and Running Elasticsearch. We recommend waiting until the end of step 5 below to deal with Elasticsearch.

These instructions will provide some guidance on installing the required dependencies and getting Arches up and running quickly.

- Installing Dependencies Ubuntu
- Installing Dependencies macOS
- Installing Dependencies Windows

Installing Arches

Once you have all of the dependencies installed, open a command prompt and use the following steps to install the latest stable release of Arches. To install the latest non-stable Arches code (generally only necessary for developers), you can follow these installation steps.

1. Create and enter a new directory called Projects:

```
mkdir Projects && cd Projects
```

2. Create a virtual environment called ENV:

```
virtualenv ENV
```

3. Activate the new virtual environment

   Linux and macOS:

```
source ENV/bin/activate
```

   Windows:

```
ENV\Scripts\activate
```

Note: At this point (once the virtual environment is activated), Windows users need to manually install Shapely. Follow these instructions.

4. Install Arches into your virtual environment

   You can install Arches with pip:

```
pip install arches --no-binary :all:
```

   Warning: Running a pip installation without the `--no-binary :all:` argument will install `arches` in your virtual environment, but will not install all of the separate python packages that you need. This will cause you to not be able to proceed with the following steps.

5. Create a new Arches project

   A project is a standalone Arches app that sits outside of your virtual environment and facilitates all of the customizations that you will need to make one installation of Arches different from the next.

   Linux and macOS:

```
arches-project create my_project
```

   Windows:

```
python C:\Projects\ENV\Scripts\arches-project create my_project
```

   Note: On Windows, the above command will end in an error on the last step of the command. The workaround for this is to enter `my_project\my_project\` (the directory that contains `bower.json`) and run this command: `bower install`.

   Warning: On Windows, you must specify the path to your GDAL library before continuing. Open `settings.py` in your new project and add this line: `GDAL_LIBRARY_PATH = 'C:/OSGeo4W64/bin/gdal201.dll'` Be sure to adjust the path as necessary for your installation, and note the *forward* slashes.

   At this point you should install ElasticSearch (unless you have done so already) before continuing.

6. Setup your Arches database

   First, make sure ElasticSearch is installed and running

   Then create your Arches database with:

```
python manage.py packages -o setup_db
```

7. Run the development server

   To view Arches, run:

```
python manage.py runserver
```

   You can now view Arches by navigating to `localhost:8000` in a browser.

More about Arches Projects

Arches Projects facilitate all of the customizations that you will need to make one installation of Arches different from the next. You can update html to modify web page branding, and add functions, datatypes, and widgets to introduce new functionality. A project sits outside of your virtual environment, and can thus be transferred to any other system where Arches is installed.

The general structure of a new Arches project is::

```
my_project/
- manage.py
- my_project/
      - settings.py
      - datatypes/
      - functions/
      - media/
      - templates/
      - widgets/
```

Note that not all files are shown

settings.py

Many project-specific settings are defined here. You will likely want to add a `settings_local.py` file that can separately store variables that you may want to keep out of the public eye (db passwords, API keys, etc.).

templates

This directory holds HTML templates that you can modify to customize the branding and general appearance of your project.

datatypes, fuctions, and widgets

These directories will store the custom datatypes, functions, and widgets that you can create for the project. Developers interested in pursuing these customizations should start with this customization documentation.

CHAPTER 3

Initial Configuration

Quick Start

Now that you have Arches installed, there are a few extra settings you must configure to make it fully operational. In the Arches interface, head to the **System Settings** menu.

The most important settings to begin with are related to the map on the Search page.

1. **Enter your Mapbox API Key.** By default, Arches uses a few basemap services from Mapbox, for which you need to provide a key. You can get a free key at mapbox.com, and for installations that do not expect exceptionally heavy traffic, this free key will be sufficient. Once you have obtained the key, copy it from Mapbox (it will start with `pk.`). Go to **System Settings – > Map Settings –> Mapbox API** and enter key there.

   If you don't want to use MapBox, you can avoid this step by loading in a different basemap and removing all of the default MapBox layers. More about loading different basemaps in the command line reference documentation.

2. **Set the default Map Zoom and Project Extent settings** The Map Zoom is useful for geometry editing, but note that the Search page will automatically zoom to the extent of your search results every time they are updated. The Project Area is very important as it defines the area for your hexagon bins. It may be best to open a new tab with your Search page, make a change here in the Settings, and then refresh your Search page to preview the changes you make.

3. **Change Hexagon Bin Settings** Finally, you can change the size and precision of the search result hexagon bins. We recommend changing these settings in small increments, as making a *small* bin size with a *large* project area (for example) can be costly for your browser and may cause it to crash when loading the Search page.

After getting the Map Settings figured out, you will probably want to change the name of your app (System Settings), or create some Saved Searches to make it easier for the users to explore your database (Saved Searches).

All System Settings

System Settings

Default Application Settings

- **Application Name** - Name of your Arches app, to be displayed in the browser title bar and elsewhere.
- **Default Data Import/Export User** - Name to associate with data that is imported into the system.

Web Analytics

If you have made a Google Analytics Key to track your app's traffic, enter it here.

Thesaurus Service Providers

Advanced users may create more SPAQRL endpoints and register them here. These endpoints will be available in the RDM and allow you to import thesaurus entries from external sources.

Map Settings

Mapbox API

Arches uses the Mapbox mapping library for map display and data creation. Arches also supports Mapbox basemaps and other services.

- **Mapbox API Key (Optional)** - By default, Arches uses some basemap web services from Mapbox. You will need to create a free API key (or "access token") for these services to be activated. Alternatively, you could remove all of the default basemaps and add your own, non-Mapbox layers.
- **Mapbox Sprites** - Path to Mapbox sprites (use default).
- **Mapbox Glyphs** - Path to Mapbox glyphs (use default).

Project Extent

Draw a polygon representing your project's extent. These bounds will serve as the default for the cache seed bounds, search result grid bounds, and map bounds in search, cards, and reports.

Map Zoom

You can define the zoom behavior of your maps by specifying max/min and default values. Zoom level 0 shows the whole world (and is the minimum zoom level). Most map services support a maximum of 20 or so zoom levels.

Search Results Grid

Arches aggregates search results and displays them as hexagons. You will need to set default parameters for the hexagon size and precision.

> **Warning:** A large project area combined with a small hexagon size and/or high precision will take a very long time to load, and can crash your browser. We suggest changing these settings in small increments to find the best combination for your project.

Basic Search Settings

Set the default search results behavior. This is also where you will define the max number of resources per export operation.

Temporal Search Settings

Arches creates a Time Wheel based on the resources in your database, to allow for quick temporal visualization and queries. A few aspects of this temporal search are defined here.

- **Color Ramp** - Currently unused (saved for future implementation). The color ramp for the time wheel. For further reference, check out the d3 API reference.
- **Time wheel configuration** - Currently unused (saved for future implementation).

Saved Searches

Arches allows you save a search and present it as convenience for your users. Saved Searches appear as search options in the main Search page. Creating a Saved Search is a three-step process.

1. **Specify Search Criteria** - Go to the Search page and enter all the criteria you would like to use to configure your Saved Search. You may notice that with the addition of each new search filter (either by using the term filter, map filtering tools, or temporal filters) the URL for the page will change.
2. **Copy the URL** - In your browser address bar, copy the *entire* URL. This will be a long string that defines each of the search filters created in step 1.
3. **Create the Saved Search** - Finally, head back to this page and fill out the settings that you see at left. You can also upload an image that will be shown along with your Search Search.

CHAPTER 4

Managing Map Layers

Different Types of Layers

Arches allows a great deal of customization for the layers on the search map. The contents of the following section will be useful when using the **Map Layer Manager** to customize your layers.

Resource Layers

Resource Layers display the resource layers in your database. One Resource Layer is created for each node with a geospatial datatype (for example, `geojson-feature-collection`). You are able to customize the appearance and visibility of each Resource Layer in the following ways.

Styling

Define the way features will look on the map. The example map has demonstration features that give you a preview of the changes you make. You can choose to use Advanced Editing to create a more nuanced style. Note that changes made in **Advanced Editing** will not be reflected if you switch back to basic editing. For styling reference, checkout the MapBox Style Specification.

Clustering

Arches uses "clustering" to better display resources at low zoom levels (zoomed out). You are able to control the clustering settings for each resource layer individually.

- Cluster Distance - distance (in pixels) within which resources will be clustered
- Cluster Max Zoom - zoom level after which clustering will stop being used
- Cluster Min Points - minimum number of points needed to create a cluster

Caching

Caching tiles will improve the speed of map rendering by storing tiles locally as they are creating. This eliminates the need for new tile generation when viewing a portion of the map that has already been viewed. However, caching is not a simple matter, and it is disabled by default. Caching is only advisable if you know what you are doing.

Basemaps and Overlays

A Basemap will always be present in your map. Arches comes with a few default basemaps, but advanced users are able to add more (see below).

Overlays are the best way to incorporate map layers from external sources. On the search map, a user is able to activate as many overlays as desired simultaneously. Users can also change the transparency of overlays. Note that Search Results and Search Markers are automatically added to the search map as overlays, and can be customized separately. New overlays can be added in the same manner as new basemaps (see below).

Styling

Note that depending on the type of layer, there are different styling options. For styling reference, checkout the MapBox Style Specification.

Settings

- Layer name - Enter a name to identify this basemap.
- Default search map - For basemaps, you can designate one to be the default. For overlays, you can choose whether a layer appears on the in the search map by default. Note that in the search map itself you can change the order of overlays.
- Layer icon - Associate an icon with this layer

Adding New Map Layers

An Arches admin can add new layers to the map by registering them through the command line interface. A newly registered layer can be designated as either a basemap or an overlay. However, because the treatment of basemaps and overlays differs only in the front-end of the app, consider the following documentation to apply equally to both.

New map layers can come from many different geospatial sources – from shapefiles to GeoTIFFs to external Web Map Services to reconfigurations of the actual resource data stored within Arches.

New map layers can be created with two general definitions, as MapBox layers or tileserver layers, each with its own wide range of options.

For working examples, please see our arches4-geo-examples repo.

MapBox Layers

```
python manage.py packages -o add_mapbox_layer -j /path/to/mapbox_style.json -n
"New MapBox Layer"
```

Arches allows you to make direct references to styles or layers that have been previously defined in MapBox Studio. You can make entirely new basemap renderings, save them in your MapBox account, then download the style definition and use it here. Read more about MapBox Styles.

Additionally, you can take a MapBox JSON file and place any mapbox.js layer definition in the `layers` section, as long as you define its source in the `sources` section.

Note: One thing to be aware of when trying to cascade a WMS through a MapBox layer is that mapbox.js is much pickier about CORS than other js mapping libraries like Leaflet. To use an external WMS or tileset, you may be better off using a tileserver layer as described below.

Tileserver Layers

```
python manage.py packages -o add_tileserver_layer -t /path/to/
tileserver_config.json -n "New Tileserver Layer"
```

Arches comes with an embedded tileserver called TileStache, which allows Arches to generate tiles internally from many different data sources. In fact, this tileserver is what creates layers from your database resources that are visible on the map.

To add a new tileserver layer, you need a .json file that contains a TileStache-compliant layer definition. Within this file, you can use any of the many different data provider classes from Tilestache. The .json file that you load into Arches for a tileserver layer should have three sections:

```
{
    "type" : This value should be "raster" or "vector".
    "layers" : This is a mapbox.js layer definition which defines the style of the␣
↪layer and links the source name with the layer name.
    "config" : This is the tileserver configuration that will be used by TileStache.␣
↪Refer to TileStache docs and place the entire "provider" section into this "config"␣
↪section.
}
```

Here's a full example of a tilestache file that makes a layer from data in PostGIS (a table called "rivers"):

```
{
    "type": "vector",
    "layers": [{
        "id": "rivers",
        "type": "line",
        "source": "rivers",
        "source-layer": "rivers",
        "layout": {
            "visibility": "visible"
        },
        "paint": {
            "line-width": 2,
            "line-color": "rgb(37, 58, 241)"
        }
    }],
    "config": {
        "provider": {
            "class": "TileStache.Goodies.VecTiles:Provider",
            "kwargs": {
                "dbinfo": {
                    "host": "localhost",
                    "user": "postgres",
                    "password": "postgis",
                    "database": "arches",
                    "port": "5432"
                },
                "simplify": 0.5,
```

```
                "queries": [
                    "select gid as __id__, name, st_asgeojson(geom) as geojson, st_
↪transform(geom, 900913) as __geometry__ from rivers"
                ]
            }
        },
        "allowed origin": "*",
        "compress": true,
        "write cache": false
    }
}
```

Tileserver Mapnik Layers

```
python manage.py packages -o add_tileserver_layer -m /path/to/mapnik_config.
xml -n "New Mapnik Tileserver Layer"
```

Mapnik is the provider that TileStache uses to serve rasters, and is very commonly used in Arches. Arches allows you to upload a Mapnik XML file to configure a new tileserver layer, instead of creating the full JSON file. This is the **easiest way to make layers from GeoTiffs and shapefiles**. A basic example of a Mapnik XML file is shown below (it points to a geotiff named `hillshade.tif`). For more about creating these XML files, see the Mapnik XML reference:

```
<Map background-color="transparent">
    <Layer name="Hillshade">
        <StyleName>raster</StyleName>
        <Datasource>
            <Parameter name="type">gdal</Parameter>
            <Parameter name="file">hillshade.tif</Parameter>
            <Parameter name="nodata">0</Parameter>
        </Datasource>
    </Layer>
    <Style name="raster">
        <Rule name="rule 1">
            <RasterSymbolizer opacity=".7" scaling="bilinear" mode="normal" />
        </Rule>
    </Style>
</Map>
```

Making Selectable Vector Layers

In Arches, it's possible to add a vector layer whose features may be "selectable". This is especially useful during drawing operations. For example, a building footprint dataset could be added as a selectable vector layer, and while creating new building resources you would select and "transfer" these geometries from the overlay to the new Arches resource.

1. First, the data source for the layer may be geojson or vector tiles. This could be a tile server layer serving vector features from PostGIS, for example.
2. Add a property to your vector features called "geojson".
3. Populate this property with either the entire geojson geometry for the feature, or a url that will return a json response containing the entire geojson geometry for the feature. This is necessary to handle the fact that certain geometries may extend across multiple vector tiles.

4. Add the overlay as you would any tileserver layer (see above).

You will now be able to add this layer to the map and select its features by clicking on them.

CHAPTER 5

Managing Resources in Arches

What Are Resources?

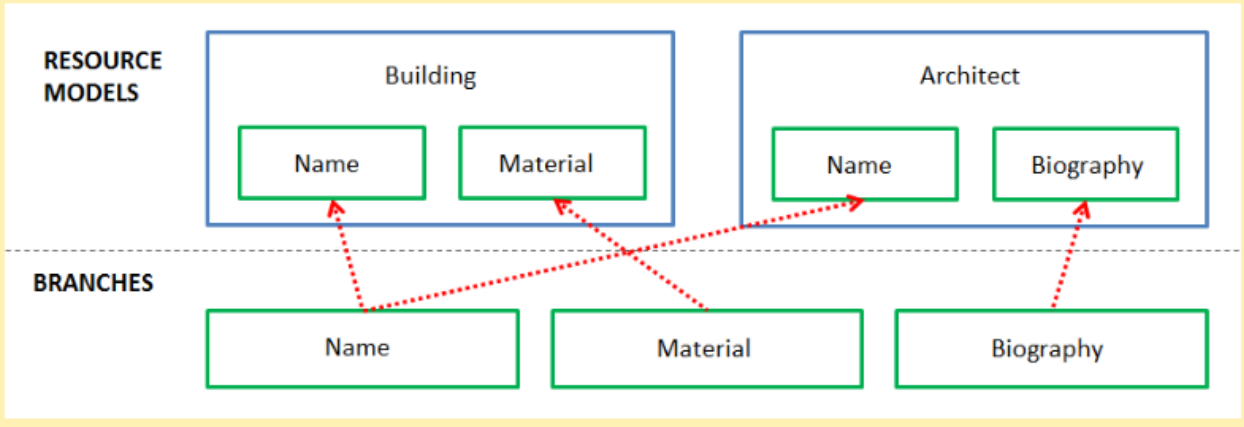
Resources are what we call "database records". Are you using Arches to create an inventory of historic buildings? In the Arches, each one of those buildings will be recorded as a resource. It can have its own name, architectural style, location, etc.

What Are Resource Models and Branches?

A central tenant of Arches' design has always been the ability to divide your inventory into distinct categories. For example, you may want to inventory buildings, as well as the architects that designed them. You would want two different categories: one called "Building" and another called "Architect". In Arches v4, we are calling these categories Resource Models. Using different Resource Models allows you to record distinct sets of information for Buildings as compared to Architects, even though all of the records are stored together in your single Arches database.

You can define these Resource Models within the Arches app itself, and to make this process easier we've added an intermediary concept called Branches. These are modular components that, after being created separately, may be added to any Resource Models you already have.

Consider the following example: You need to create a Building Resource Model and an Architect Resource Model. For resources of both types you will need to record a name. However, for the buildings you'd like to record building materials, while the architects should get some biographical information. Thus, we are talking about two new Resource Models (Building, Architect) and three new Branches (Name, Materials, Biography). You will make all five entities, and then add the appropriate Branches to the appropriate Resource Models.



Managing Resource Models and Branches happens in the Arches Designer, and to enter this part of the system you must have the appropriate permissions. Once you have logged in, choose MANAGE DATA from the Arches home page, and find the Arches Designer in the left side-bar menu. You are now ready to create, modify, and remove Resource Models and Branches.

CHAPTER 6

Using the Arches Designer

Introduction

The Arches Designer is a set of tools that give you full control over the creation of new Resource Models. This section serves as a reference manual for the various components of the Arches designer.

Note: Most of this section's content can be found directly within the Arches app. Just click ? in the upper right corner of any page.

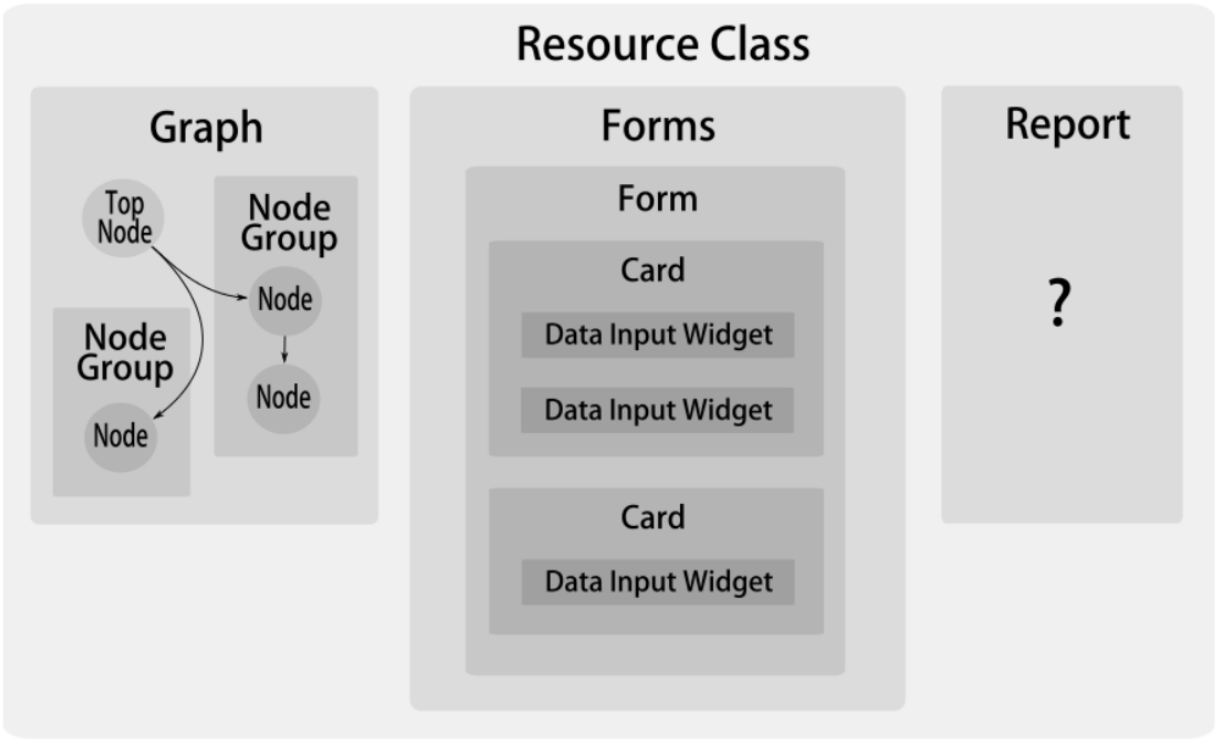
- *Branch/Resource Model Settings*
- *Graph Designer*
- *Card Designer*
- *Form Designer*
- *Report Designer*
- *Function Designer*

Overview

(old content, may be more appropriate elsewhere. image text needs updating)

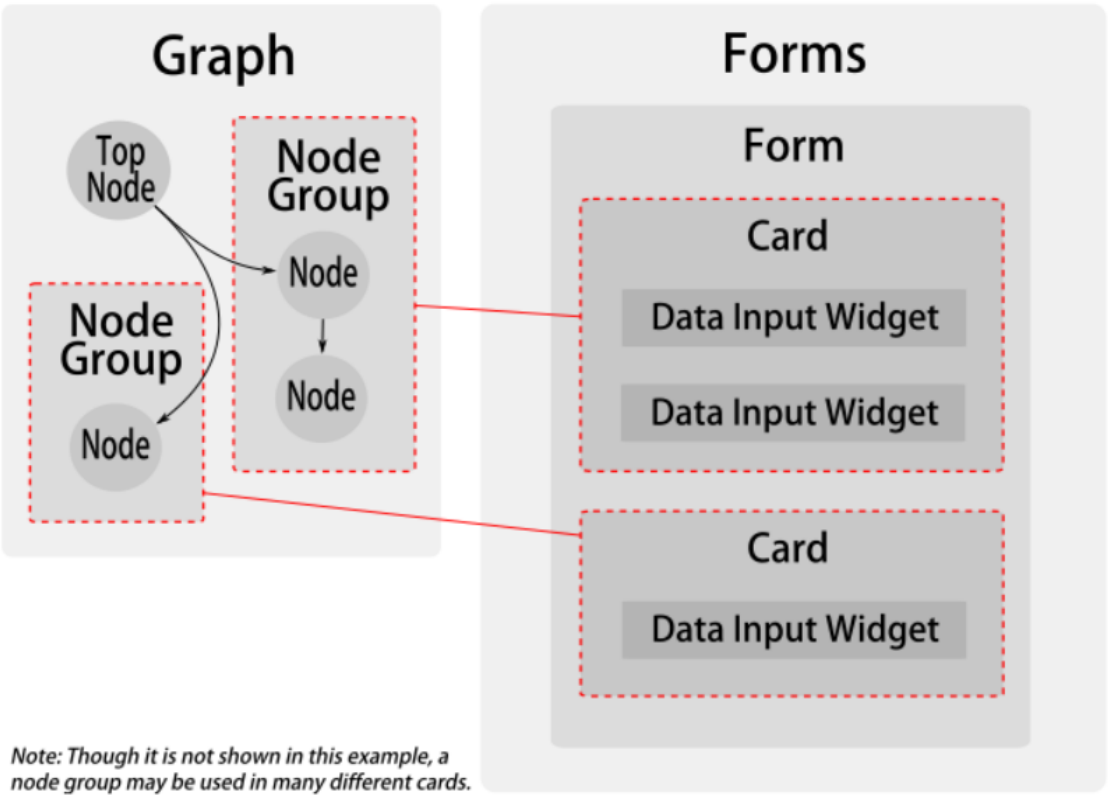
Resources in an Arches database are separated into distinct resource "classes," and in v4 the way a user creates and manages these classes has been completely redesigned. Arches now handles all of these operations in the Resource Manager (which you must be logged in to access), and resource classes are made of modular components that can be re-used throughout the system.

The basic components of a resource class are a graph, a series of data entry forms, and a report. In the Arches interface, each of these components comprise various sub-components, as illustrated below.



All of these UI components work together to facilitate data input and display:

The graph is made of node groups that define what information will be gathered for a given resource class. The forms are made of cards that are tied to specific node groups and define what input widget will be used to gather a value for each node. Behind the scenes, these components are defined using many Python classes that are stored as Django models. Instances of these classes are stored in separate database tables, in order to support and enforce relationships. For example, the following the following illustrates a very basic example of the interaction between node groups and cards.



Once values have been entered into the widgets in a card, the business data is saved as a "tile". These tiles are stored as JSON objects in the Postgres database, and are the only format in which resource data exist.

In the example, two very simple node groups are shown. Note that a node group may in fact be very complex, and hold many more than two nodes. As you can see, these node groups are used as the basis for "cards", which are used in the data entry forms. Each card is associated with one or more node groups, and assigns a widget (e.g. a dropdown selection list or calendar date picker) to each node. A card can also record other properties like validation functions.

Branch/Resource Model Settings

<pull contents from the html help templates>

Graph Designer

<pull contents from the html help templates>

Note: One should refer to CIDOC CRM (http://www.cidoc-crm.org/) to make an informed decision about what class or property to assign to nodes and edges respectively.

Card Designer

<pull contents from the html help templates>

Form Designer

Report Designer

<pull contents from the html help templates>

Function Designer

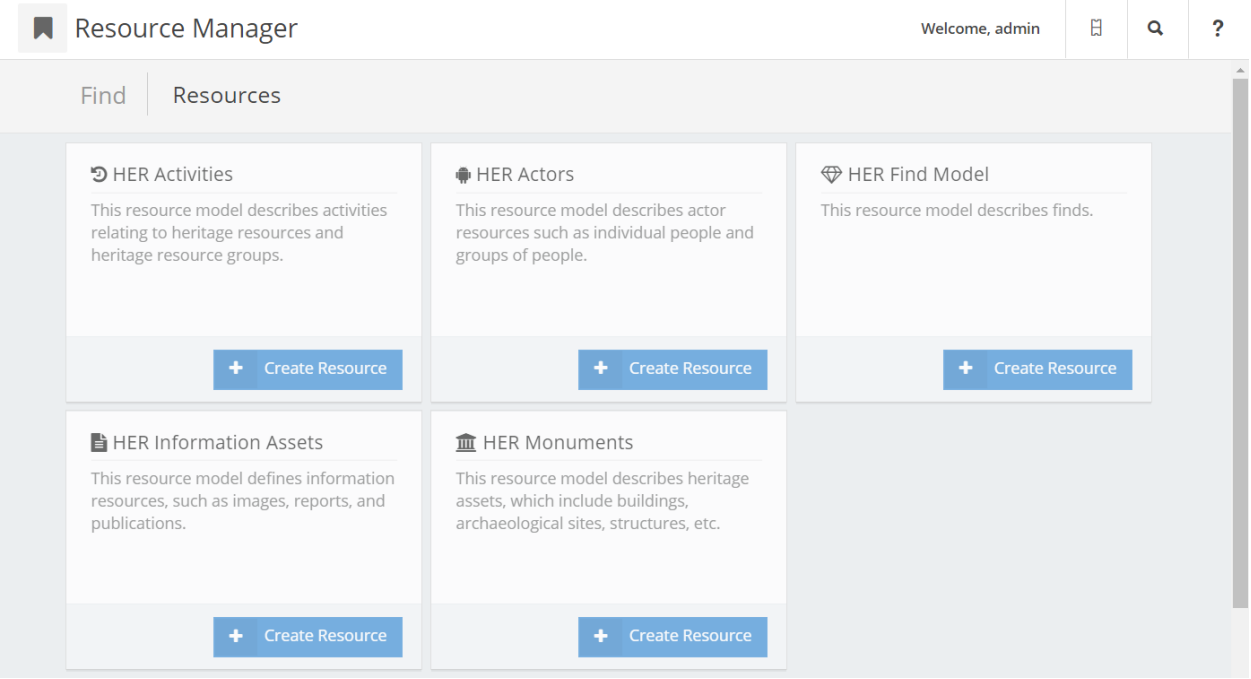
<pull contents from the html help templates>

CHAPTER 7

Creating Resources

Resource Manager

You may create new Resources only if you have access to the Resource Manager page. From there, you will begin by choosing which Resource Model you would like to use. Note that a Resource Model must have its status set to **active** and must have **at least one menu** added it in order for it to be used.



Your Resource Manager page may look different, depending on what Resource Models you have set up in your database.

Resource Editor

The Resource Editor is used to create new or edit existing Resources. On the left-hand side of the page you are presented with the list of Menus associated with this Resource Model, each holding one or more data-entry cards. Cards may be further divided, with tabs appearing across the top.

The procedure for entering data depends on the widget: sometimes you'll enter text, sometimes select from a dropdown menu, or sometimes record a location by drawing geometry on a map. One thing to keep in mind generally is that certain attributes are allowed to have multiple values recorded, while some will only allow one value.

As an example, Name/Name Type pairs tend to allow multiple values. Above, two new Name/Name Type pairs are added. Note the manner in which saved values are summarized above the data entry widget. You may directly edit these existing values where they appear in this summary.

Related Resources

In the Resource Editor you can also access the Related Resources page. Creating a resource relation is a three-step process.

1. Find a Resource

   Using one (or more) of the available search methods, find the resource to which you would like to create a relation. As you apply more filters, the search results will be updated.

2. Create the Relation

   Once the resource you would like to relate to appears in the search results, click "Relate Resource" to create the relation. You will see the new relation visualized on the right. You can view existing relations as a grid (table) or as a graph (connected nodes). *If you see "Cannot Be Related" next to a Resource in the search results, this means that relationships have not been enabled between these two Resource Models. A user with access to the Arches Designer can change this by editing the settings for either Resource Model.*

3. Edit Relation Properties

   After a relation has been created, you can further refine its properties, such as what type of relation it is, how long it lasted, etc. While viewing the relation in grid mode, begin by selecting the relation in the table. You will see the "Delete Selected" button appear. Next click "relation properties", enter the information, and don't forget to "Save" when finished.

$ virtualenv env $ source env/bin/activate (env)$ cd arches

Install arches and create a new project (env)$ python setup.py install #Later this will be *pip install arches* (env)$ cd [to where you want to create your project] (env)$ arches-project create mynewproject (env)$ cd mynewproject

In your settings use the default arches settings for your database and elasticsearch ports. If you want to run more that one project on a machine, you will have to modify these to avoid conflicts

install and run elasticsearch (env)$ python manage.py es install (env)$ ./elasticsearch-5.2.1/bin/elasticsearch -d

set up your database, import graphs and start your django server (env)$ python manage.py packages -o setup_db (env)$ python manage.py packages -o import_graphs (env)$ python manage.py runserver

to add a function ... #list the names of functions you have registered: (env)$ python manage.py fn list

#register a new function (-s [the path to your function's py file]): (env)$ python manage.py fn register –source /Users/me/Documents/projects/mynewproject/mynewproject/functions/sample_function.py

#navigate to your graph's function manager to confim that your new function is there and working. If it's not, you may want to unregister your function and make additional changes. #to unregister your function, simply run: (env)$ python manage.py fn unregister –name 'Sample Function'

to add a function ... #list the names of functions you have registered: (env)$ python manage.py fn list

#register a new function (-s [the path to your function's py file]): (env)$ python manage.py fn register –source /Users/me/Documents/projects/mynewproject/mynewproject/functions/sample_function.py

#to unregister your function, simply run: (env)$ python manage.py fn unregister –name 'Sample Function'

to add a datatype ... #list the names of datatypes you have registered: (env)$ python manage.py fn list

#register a new datatype (-s [the path to your datatype's py file]): (env)$ python manage.py datatype register –source /Users/me/Documents/projects/mynewproject/mynewproject/datatypes/wkt_point.py

#make changes to your datatype's properties (-s [the path to your datatype's .py file]): (env)$ python manage.py datatype update –source /Users/me/Documents/projects/mynewproject/mynewproject/datatypes/wkt_point.py

#to unregister your datatype, simply run: (env)$ python manage.py datatype unregister -d 'wkt-point'

to add a widget ... #list the names of widgets you have registered: (env)$ python manage.py fn list

#register a new widget (-s [the path to your widget's .json file]): (env)$ python manage.py widget register –source /Users/me/Documents/projects/mynewproject/mynewproject/widgets/wkt_point.json

#make changes to your widget's properties (-s [the path to your widget's .json file]): (env)$ python manage.py widget update –source /Users/me/Documents/projects/mynewproject/mynewproject/widgets/wkt_point.json

#to unregister your widget, simply run: (env)$ python manage.py widget unregister -n 'wkt-point-widget'

CHAPTER 8

Resource Import/Export

Currently, all data import and export operations happen through the Arches command line interface.

Uploading a CSV to Arches

To do a bulk import of resources to your Arches database, you must create a correctly formatted CSV (comma-separated values) file. We recommend MS Excel or Open Office for this task. More advanced users will likely find a custom scripting effort to be worthwhile.

To upload a shapefile see Uploading a Shapefile to Arches below.

All values containing commas must be quoted so the value is not misinterpreted by the importer as multiple csv columns. This includes strings, concept-lists, and multi-geoms.

The workflow for creating a CSV should be something like this:

1. Identify which Resource Model you are loading data into
2. Download the **mapping file** and **concepts file** for that resource model
3. Modify the mapping file to reference your CSV
4. Populate the CSV with your data
5. Import the CSV using the import_business_data command.

CSV File Requirements

Each row in the CSV can contain the attribute values of one and only one resource.

The first column in the CSV must be named `ResourceID`. ResourceID is a user-generated unique ID for each individual resource. If ResourceID is a valid UUID, Arches will adopt it internally as the new resource's identifier. If ResourceID is not a valid UUID Arches will create a new UUID and use that as the resource's identifier. Subsequent columns can have any name.

ResourceIDs must be unique among all resources imported, not just within each csv, for this reason we suggest using UUIDs.

Resource ID	attribute 1	attribute 2	attribute 3
1	attr. 1 value	attr. 2 value	attr. 3 value
2	attr. 1 value	attr. 2 value	attr. 3 value
3	attr. 1 value	attr. 2 value	attr. 3 value

Simple CSV with three resources, each with three different attributes.

Or, in a raw format (if you open the file in a text editor), the CSV should look like this:

```
Resource ID,attribute 1,attribute 2,attribute 3
1,attr. 1 value,attr. 2 value,attr. 3 value
2,attr. 1 value,attr. 2 value,attr. 3 value
3,attr. 1 value,attr. 2 value,attr. 3 value
```

Multiple lines may be used to add multiple attributes to a single resource. You must make sure these lines are contiguous, and every line must have a ResourceID. Other cells are optional.

Resource ID	attribute 1	attribute 2	attribute 3
1	attr. 1 value	attr. 2 value	attr. 3 value
2	attr. 1 value	attr. 2 value	attr. 3 value
2		attr. 2 additional value	
3	attr. 1 value	attr. 2 value	attr. 3 value

CSV with three resources, one of which has two values for attribute 2.

Depending on your Resource Model's graph structure, some attributes will be handled as "groups". For example, `Name` and `Name Type` attributes would be a group. Attributes that are grouped must be on the same row. However, a single row can have many different groups of attributes in it, but there may be only one of each group type per row. (e.g. you cannot have two names and two name types in one row).

Resource ID	name	name_type	description
1	Yucca House	Primary*	"this house, built in..."
2	Big House	Primary*	originally a small cabin
2	Old Main Building	Historic*	
3	Writer's Cabin	Primary*	housed resident authors

CSV with three resources, one of which has two groups of `name` *and* `name_type` *attributes. See below for more on the 'Primary' and 'Historic' labels.*

You must have values for any required nodes in your resource models.

The data_type attribute in the mapping file(see below) can help you determine what data type (string, number, conceptid, geometry, etc.) is valid for each column in your csv.

Some data types require specially formatted values:

- **Dates** All dates must be formatted as YYYY-MM-DD. Dates that are not four digits must be zero padded.
- **Geometry** All geometry must be formatted in "well-known text" (WKT), and that all coordinates stored as WGS 84 (EPSG:4326) decimal degrees. Multi geoms must be in quotes.
- **Rich Text** Rich text must be quoted html.
- **Concepts** See the *Concepts* section below.
- `concept-list` and `domain-value-list` Values must be represented as quoted, comma-separated strings in the source CSV file. Each value within the concept/domain list does not need to be in quotation marks.

  `concept-list` example:

```
"3b4bddbb-0ca6-44d5-8622-ce9982412883,10ac2b9b-3425-4c3c-9b5c-
↪983bb16f93d3,052e8714-3f19-4179-b1f9-dcbe45df673e"
```

Mapping File Requirements

All CSV files must be accompanied by a **mapping file**. This is a JSON-structured file that indicates which node in a Resource Model's graph each column in the CSV file should map to. The mapping file should contain the source column name populated in the `file_field_name` property for all nodes in a graph the user wishes to map to. The mapping file should be named exactly the same as the CSV file but with the extension '.mapping', and should be in the same directory as the CSV.

To create a mapping file for a Resource Model in your database, go to the Arches Designer landing page. Find the Resource Model into which you plan to load resources, and choose Export Mapping File from the Manage menu.

Unzip the download, and you'll find a `.mapping` file as well as a `...concepts.json` file (more on the latter below). The contents of the mapping file will look something like this:

```
{
    "resource_model_id": "bbc5cee8-fa16-11e6-9e3e-026d961c88e6",
    "resource_model_name": "HER Buildings",
    "nodes": [
        {
            "arches_nodeid": "bbc5cf1f-fa16-11e6-9e3e-026d961c88e6",
            "arches_node_name": "Name",
            "file_field_name": "",
            "data_type": "concept",
            "concept_export_value": "label",
            "export": false
        },
        {
            "arches_nodeid": "d4896e3b-fa30-11e6-9e3e-026d961c88e6",
            "arches_node_name": "Name Type",
            "file_field_name": "",
            "data_type": "concept",
            "concept_export_value": "label",
            "export": false
        },
        ...
    ]
}
```

The mapping file contains cursory information about the resource model (name and resource model id) and a listing of the nodes that comprise that resource model. Each node contains attributes to help you import your business data (not all attributes are used on import, some are there simply to assist you). The concept_export_value attribute is only present for nodes with datatypes of *concept*, 'concept-list', 'domain', and 'domain-list' - this attribute is not used for import. It is recommended that you not delete any attributes from the mapping file. If you do not wish to map to a specfic node simply leave the `file_field_name` attribute blank.

You will now need to enter the column name from your CSV into the `file_field_name` in appropriate node in the mapping file. For example, if your CSV has a column named "activity_type" and you want the values in this column to populate "Activity Type" nodes in Arches, you would add that name to the mapping file like so:

```
{
    ...
```

```
        {
            "arches_nodeid": "bbc5cf1f-fa16-11e6-9e3e-026d961c88e6",
            "arches_node_name": "Activity Type",
            "file_field_name": "activity_type", <-- place column name here
            "data_type": "concept",
            "concept_export_value": "label",
            "export": false
        },
    ...
}
```

To map more than one column to a single node, simply copy and paste that node within the mapping file.

Concept Values in CSVs, and the concepts file

Values in the CSV that represent concepts in the RDM (Reference Data Manager) must not use the label of that concept, but its actual valueid that represents that concept in the RDM. Thus, in the above example, 'Primary' and 'Historic', because these are actually concepts stored in the RDM, would actually need to be replaced by the valueid (a UUID) for their respective concepts.

To aid with the process, a **concepts file** is created every time you download a mapping file, which lists the concept valueids and corresponding labels for all of the Concept Collections associated with any of the Resource Model's nodes. For example:

```
"Name Type": {
    "ecb20ae9-a457-4011-83bf-1c936e2d6b6a": "Historic",
    "81dd62d2-6701-4195-b74b-8057456bba4b": "Primary"
},
```

Thus, using the concepts file, the example above should be changed to look like this:

Resource ID	name	name_type	description
1	Yucca House	81dd62d2-6701-4195-b74b-8057456bba4b	"this house, built in..."
2	Big House	81dd62d2-6701-4195-b74b-8057456bba4b	originally a small cabin
2	Old Main Building	ecb20ae9-a457-4011-83bf-1c936e2d6b6a	
3	Writer's Cabin	81dd62d2-6701-4195-b74b-8057456bba4b	housed resident authors

Uploading a Shapefile to Arches

Uploading a shapefile to Arches is very similar to uploading a csv file with a few exceptions. The same rules apply to rich text, concept data, grouped data, and contiguousness. And, like csv import, shapefile import requires a mapping file.

The shapefile must contain a field with a unique identifier for each resource named 'ResourceID'.

The shapefile must be in WGS 84 (EPSG:4326) decimal degrees.

The shapefile must consist of at least a .shp, .dbf, .shx, and .prj file. It may be zipped or unzipped.

Dates in a shapefile can be in Esri Shapefile date format, Arches will convert them to the appropriate date format.

In your mapping file, the node you wish to map the geometry to must have a `file_field_name` value of 'geom'.

To import a shapefile use the same command as csv import, simply point to your shapefile instead of your csv file. import_business_data

Note: More complex geometries may encounter a `mapping_parser_exception` error. This error occurs when a geometry is not valid in elasticsearch. To resolve this, first make sure your geometry is valid using ArcMap, QGIS, or PostGIS. Next, you can modify the precision of your geometry to 5 decimals or you can simplify your geometry using the QGIS simplify geometry geoprocessing tool, or the PostGIS st_snaptogrid function.

Exporting Arches Data

Currently all export must happen through the command line interface. Alternatively, users with database access have the option of writing a resource view, adding the view to a GIS client, and exporting data from there. Export commands are listed in the Wiki documentation.

Resource Database Views

To export to tabular formats such as shapefile, it is necessary to flatten the graph structure of your resources. One way to do this is to create a database view of your resource models. Arches does not do this automatically because there are many ways to design a flattened table depending on your needs.

You can add any number of database views representing a given resource model either for export, or to connect directly to a GIS client such as QGIS or ArcGIS. When writing a view to support shapefile export be sure that your view does not violate any shapefile restrictions. For example, shapefile field names are limited to 10 characters with no special characters and text fields cannot store more than 255 characters.

If you plan to use the arches *export* command to export your view as a shapefile, you also need to be sure that your view contains 2 fields: *geom* with the geometry representing your resource instance's location and *geom_type* with the postgis geometry type of your *geom* column.

To write your view, you should start by getting a mapping file for your resource. You can do that by going to the Arches Designer page and then in the *manage* dropdown of your resource model select *Create Mapping File*. A zip file will be downloaded and within that file you will find your *.mapping* file. This file lists all the ids that you will need to design your view.

Below is an example of a simple resource model view. If a resource instance has a tile with geojson saved to it, that tile will be represented as a record in the view along with the corresponding nodeid and tileid. A unique id (gid) is assigned to each row. If a node has more than one geometry, the geometries are combined into a multipart geometry. If a node has more than one geometry of different types, a record will be created for each type. The UUID (ab74af76-fa0e-11e6-9e3e-026d961c88e6) in the last line of this this example is the id of the view's resource model.

1. When creating your own view, you will need to replace this UUID with your own resource model's id. You can find this UUID in your mapping file assigned to the property: *resource_model_id*.

```
CREATE OR REPLACE VIEW vw_monuments_simple AS
WITH mv AS (SELECT tileid, resourceinstanceid, nodeid, ST_Union(geom) as␣
↪geom, ST_GeometryType(geom) AS geom_type
FROM mv_geojson_geoms
GROUP BY tileid, nodeid, resourceinstanceid, ST_GeometryType(geom))
SELECT row_number() OVER () AS gid,
    mv.resourceinstanceid,
    mv.tileid,
    mv.nodeid,
    ST_GeometryType(geom) AS geom_type,
    geom
FROM mv
WHERE (SELECT graphid FROM resource_instances WHERE mv.resourceinstanceid␣
↪= resourceinstanceid) = 'ab74af76-fa0e-11e6-9e3e-026d961c88e6'
```

2. Here is a more complete example which includes columns with tile data:

```
CREATE OR REPLACE VIEW vw_monuments AS
WITH mv AS (select tileid, resourceinstanceid, nodeid, ST_Union(geom) AS␣
↪geom, ST_GeometryType(geom) AS geom_type
FROM mv_geojson_geoms
GROUP BY tileid, nodeid, resourceinstanceid, ST_GeometryType(geom))
SELECT
    row_number() over () AS gid,
    mv.resourceinstanceid,
    mv.tileid,
    mv.nodeid,
    ST_GeometryType(geom) AS geom_type,
    name_tile.tiledata ->> '677f303d-09cc-11e7-9aa6-6c4008b05c4c' AS name,
    (SELECT value FROM values WHERE cast(name_tile.tiledata ->> '677f39a8-
↪09cc-11e7-834a-6c4008b05c4c' AS uuid) = valueid ) AS nametype,
    (SELECT value FROM values WHERE cast(component.tiledata ->>'ab74b009-
↪fa0e-11e6-9e3e-026d961c88e6' AS uuid) = valueid ) AS construction_type,
    array_to_string((select array_agg(v.value) FROM unnest(ARRAY(SELECT␣
↪jsonb_array_elements_text(component.tiledata -> 'ab74afec-fa0e-11e6-
↪9e3e-026d961c88e6'))::uuid[]) item_id LEFT JOIN values v ON v.
↪valueid=item_id), ',') AS const_tech,
    (SELECT value FROM values WHERE cast(record.tiledata ->> '677f2c0f-
↪09cc-11e7-b412-6c4008b05c4c' AS uuid) = valueid ) AS record_type,
    geom
FROM mv
LEFT JOIN tiles name_tile
    ON mv.resourceinstanceid = name_tile.resourceinstanceid
    AND name_tile.tiledata->>'677f39a8-09cc-11e7-834a-6c4008b05c4c'
    != ''
LEFT JOIN tiles component
    ON name_tile.resourceinstanceid = component.resourceinstanceid
    AND component.tiledata->>'ab74afec-fa0e-11e6-9e3e-026d961c88e6'
    != ''
LEFT JOIN tiles record
    ON name_tile.resourceinstanceid = record.resourceinstanceid
    AND record.tiledata->>'677f2c0f-09cc-11e7-b412-6c4008b05c4c'
    != ''
WHERE (SELECT graphid FROM resource_instances WHERE mv.resourceinstanceid␣
↪= resourceinstanceid) = 'ab74af76-fa0e-11e6-9e3e-026d961c88e6'
```

3. You will notice that for each node added as a column in the table, we perform a LEFT JOIN to the tiles table and the nodeid from which we want data. Here is an example joining to the tile containing the *record* node which has a nodeid of *677f2c0f-09cc-11e7-b412-6c4008b05c4c*.

```
LEFT JOIN tiles record
    ON name_tile.resourceinstanceid = record.resourceinstanceid
    AND record.tiledata->>'677f2c0f-09cc-11e7-b412-6c4008b05c4c'
    != ''
```

4. We can then define a field be referencing that tile:

```
(SELECT value FROM values WHERE cast(record.tiledata ->> '677f2c0f-09cc-
↪11e7-b412-6c4008b05c4c' AS uuid) = valueid ) AS record_type
```

5. How you define your fields depends largely on what the node datatype is:

   A node with a string datatype:

```
name_tile.tiledata ->> '677f303d-09cc-11e7-9aa6-6c4008b05c4c' AS name
```

A node with a concept value id. The following returns the concept values label:

```
(SELECT value FROM values WHERE cast(name_tile.tiledata ->> '677f39a8-
↪09cc-11e7-834a-6c4008b05c4c' AS uuid) = valueid ) AS nametype
```

A node with a concept-list. The following returns a concatenated string of concept value labels:

```
array_to_string((SELECT array_agg(v.value) FROM unnest(ARRAY(SELECT jsonb_
↪array_elements_text(component.tiledata -> 'ab74afec-fa0e-11e6-9e3e-
↪026d961c88e6'))::uuid[]) item_id LEFT JOIN values v ON v.valueid=item_
↪id), ',') AS const_tech
```

CHAPTER 9

Ontologies in Arches

Arches data is modeled with graphs. A graph is a collection of nodes, structured like branches, all emanating from the root node, which represents the resource itself. If you are modeling a building resource, you may have a root node called "Building" with a node attached to it called "Name". You can imagine that complex and thoroughly documented resources will have many, many nodes.

An ontology is a set of rules that categorizes these nodes into classes, and dictates which classes can be connected to each other. It's a "rulebook" for graph construction. Generally, this is known as a CRM (Conceptual Reference Model), and Arches comes preloaded with the CIDOC CRM v6.2, an ontology created by ICOM specifically to describe cultural heritage data. To learn more about the CIDOC CRM, visit cidoc-crm.org or view a full list of classes and properties.

When creating Resource Models and Branches, users have the option of enforcing an ontology throughout the graph, or creating a graph with no ontology. If an ontology is chosen, the Graph Designer will enforce all of the applicable node class (CRM Entities) and edge (CRM Properties) rules during use of the Graph Designer. Importantly, if a Resource Model uses an ontology one can only add Branches to it that have been made with the same ontology.

CHAPTER 10

Reference Data Manager (RDM)

The Arches Reference Data Management (RDM) tool is a core Arches module which enables the creation and maintenance of controlled vocabularies for use in dropdowns and controlled fields within the various Arches Resource forms.

The use of the RDM is restricted to the Reference Data Manager, the person responsible for maintaining the controlled vocabularies. It allows for the creation, update, amendment and deletion of concept schemes (controlled vocabularies). In addition the RDM enables you to export your schemes as SKOS-Compliant XML files as well as the import of external thesauri. For more information on SKOS see http://www.w3.org/2004/02/skos/.

Concept Schemes

A concept scheme can be viewed as an aggregation of one or more concepts and the semantic relationships (links) between those concepts.

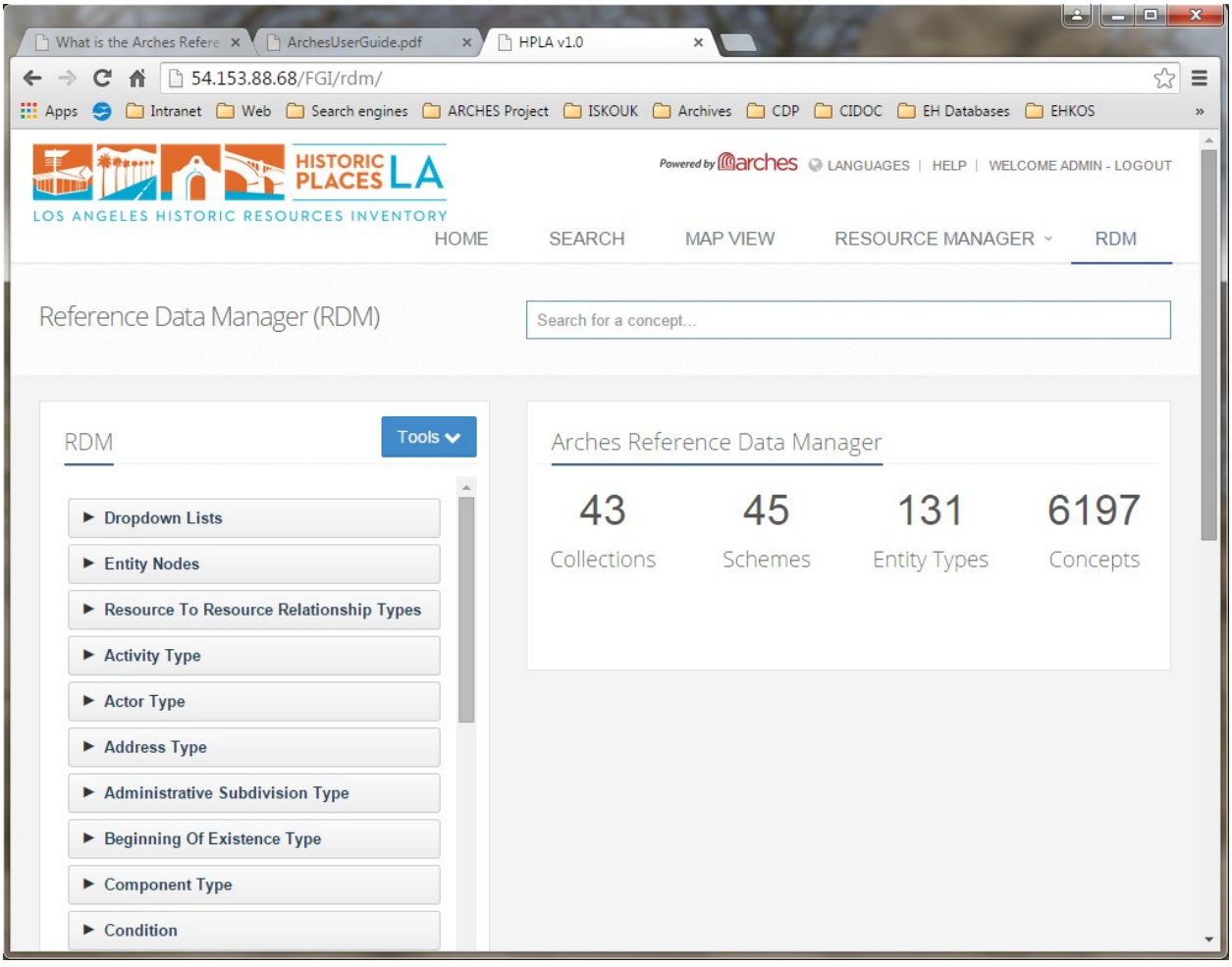
Each controlled vocabulary within the Arches RDM, whether it is a simple wordlist or a polyhierarchical thesaurus, is defined as a concept scheme. [More detail about concept schemes needed here]

Getting started

In this section you will learn about:

- **Adding a new concept scheme**
  - Adding a label to a scheme
  - Adding a note to a scheme
- **Building the scheme**
  - Adding a Top Concept to a scheme
  - Importing a Top Concept from an external scheme
  - Adding a child concept

  - Importing a child concept
  - Adding an additional Parent Concept (polyhierarchy)
  - Browsing the scheme using the graph interface
  - Adding a Related Concept
  - Adding an image to a concept
  - Searching for a concept
  - Deleting a concept
- Importing a scheme
- Exporting a scheme
- Deleting a scheme



Adding a new concept scheme

1. In the left hand panel select **Add Scheme** from the **Tools** dropdown. The *Add Concept Scheme* pop-up will appear.
2. Insert the title [Test Scheme] of the new concept scheme in the *ConceptScheme Name* field.

3. Add a brief description of the Concept Scheme in the *Scope Note* field.
4. Select the language of the ConceptScheme by clicking in the '**Language**' field. This currently defaults to *en-(US) (English)*
5. Click the **Save Changes** button. The new concept scheme will appear in the **ConceptSchemes** panel.

Adding a label to a scheme

It is possible to add multiple labels to a scheme. This is useful as some schemes may have been referred to by different names previously.

1. Select the **Test Scheme** from the left hand **RDM** panel. The *Test Scheme* will appear in the right hand panel.
2. Select **Add Label**. The *Add Concept Label* pop-up will appear.
3. Click in the field marked '**prefLabel**'. The list of label types will appear.
4. Select the label type.
5. Select the language of the label by clicking in the *Language* field. This currently defaults to *en-(US) (English)*
6. Click the **Save** button. The new label will appear in the **Labels** panel.

Adding a note to a scheme

It is possible to add multiple notes to a scheme. This allows the reference data manager to add more information regarding the scheme including the scope of what it covers, it's definition, changes to the history of the scheme, and how it should be used.

1. Select **Add Note**. The *Add Concept Note* pop-up will appear.
2. Enter the text for the new note in the '**Note Editor**' field.
3. Click in the field marked '**scopeNote**'. The list of Note types will appear.
4. **Select the relevant Note type. NB** Only one note of each type is allowed.
5. Select the language of the Note by clicking in the '**Language**' field. This currently defaults to *en-(US) (English)*
6. Click the **Save** button. The new Note will appear in the **Notes** panel.

Building the Scheme

Having created the new scheme you should now add the Top Concepts. These will form the framework for the vocabulary and act as the parents for more detailed concepts. This multi-level construction is known as the hierarchy. In a simple wordlist there will by only one level of concepts but in a complex thesaurus the hierarchy can be many levels deep.

Adding a Top Concept to a scheme

1. In the Right hand panel select **Add Top Concept** from the **Manage** dropdown. The *Add Concept* pop-up will appear.
2. Enter the text for the label in the '**Label**' field.
3. Enter the definition of the concept in the **Note**' field. The list of Note types will appear.
4. Select the language of the Note by clicking in the '**Language**' field. This currently defaults to *en-(US) (English)*

5. Select *hasTopConcept* from the '**Relation from Parent**' field.
6. Click the **Save Changes** button. The new concept will appear in the **Broader/Narrower Concepts** panel.

It may be that other concept schemes similar to the one you are developing may already exist. If this is the case it is possible to import concepts along with their attributes from an external source. By default the RDM can import concepts from the Getty Art and Architecture Thesaurus

Importing a Top Concept from an external scheme

1. In the Right hand panel select **Import Top Concept from SPARQL** from the **Manage** dropdown. The *Import Concept* pop-up will appear.
2. Select **Getty AAT** from the list of **Schemes** available.
3. In the '**Search for a concept**' field type the text of a concept, eg. houses. A selection of concepts matching the text will appear.
4. Select the appropriate concept. The *Concept Identifier* field will be populated with the URI of the concept.
5. Click the **Import** button. The new concept will appear in the **Broader/Narrower Concepts** panel.
6. Click on the concept. The Concept details panel will appear and the **Notes** panel will be populated with the external concept's *scopeNote*

Adding a child concept

1. Select the concept, which will act as the parent for the new child concept, by clicking on it. The Concept details panel will appear
2. In the Right hand panel select **Add Child** from the **Manage** dropdown. The *Add Concept* pop-up will appear.
3. Enter the text for the label in the '**Label**' field.
4. Enter the definition of the concept in the **Note**' field. The list of Note types will appear.
5. Select the language of the Note by clicking in the '**Language**' field. This currently defaults to *en-(US) (English)*
6. Select *narrower* from the '**Relation from Parent**' field.
7. Click the **Save Changes** button. The new concept will appear in the **Broader/Narrower Concepts** panel.
8. Click on the new concept. The Concept details panel will appear and the **Notes** panel will be populated with the concept's *scopeNote*

Importing a child concept

1. Select the concept, which will act as the parent for the new child concept, by clicking on it. The Concept details panel will appear.
2. In the Right hand panel select **Import Child from SPARQL** from the **Manage** dropdown. The *Import Concept* pop-up will appear.
3. Select **Getty AAT** from the list of **Schemes** available.
4. In the '**Search for a concept**' field type the text of a concept, eg. castle. A selection of concepts matching the text will appear.
5. Select the appropriate concept. The *Concept Identifier* field will be populated with the URI of the concept.

6. Click the **Import** button. The new concept will appear in the **Broader/Narrower Concepts** panel.
7. Click on the concept. The Concept details panel will appear and the **Notes** panel will be populated with the external concept's *scopeNote*

Adding an additional Parent Concept (polyhierarchy)

Some concepts may have more than one parent for example a castle is a type of fortification but it is also a domestic building. This situation where there are more than one possible parent concepts is called polyhierarchy.

1. Select the concept, which you want to add a parent concept to, by clicking on it.
2. Select **Manage Parents** from the **Manage** dropdown. The *New Parent Concept* pop-up will appear.
3. In the '**Search for a concept**' field type the text of the parent concept you are going to add, eg. domestic buildings. A selection of concepts matching the text will appear.
4. Select the appropriate concept.

Browsing the Scheme using the graph interface

For any Concept the *Broader/Narrower Concepts* panel defaults to the tree view and shows a concept's immediate broader (parent) and narrower (child) concepts. The scheme may also be browsed using the graph interface.

1. In the Broader/Narrower Concepts panel click **Show graph**. The graph view will appear centred on the concept you have chosen.
2. Navigate the graph by clicking on the 'nodes' (the circles). Clicking on a node will bring up a dialog box with the concept label and a 'x' symbol.
3. Click on the label to jump to the details for a concept

Adding a Related Concept

As part of a thesaurus it is possible to relate concepts which are not hierarchically related but may be of interest to a user. This 'Associative' relationship can be made by relating one concept to many others.

1. In the Right hand panel click on **Add Related Concept** in the *Related Concepts* panel. The *Manage Related Concepts* pop-up will appear.
2. Enter the text for the related concept in the '**Select a concept**' field. A selection of concepts matching the text will appear.
3. Select the appropriate concept.
4. Click in the **Relation type** field. The Relation Type dropdown will appear.
5. Select '**Related**'.
6. Click the **Save** button. The related term is added to the concept.

Adding an image to a concept

Searching for a concept

Deleting a concept

Deleting a concept is simple in Arches but car should be taken that the concept has not been used in any recording forms. If a concept has been used a warning message will appear informing the Reference Data Manager that all instances of the concept in use must be replaced with an alternative concept before the concept can be deleted. If the Reference Data Manager is certain the concept has not been used then the concept may be deleted using either of the following methods.

Method 1: Tree view

1. Identify the concept's parent concept and bring up its details.
2. In the *Broader/Narrower Concepts* panel make sure the tree view is visible (This is the default view).
3. Click on the '**x**' symbol next to the concept to be deleted. The *Delete a Concept* pop-up will appear.

   NB A warning message 'By deleting this concept, you will also be deleting the following concepts as well. This operation cannot be undone.' will appear. If you do not want to delete the concept click the **No** button.
4. Click the **Yes** button. The concept is deleted (along with any of its children).

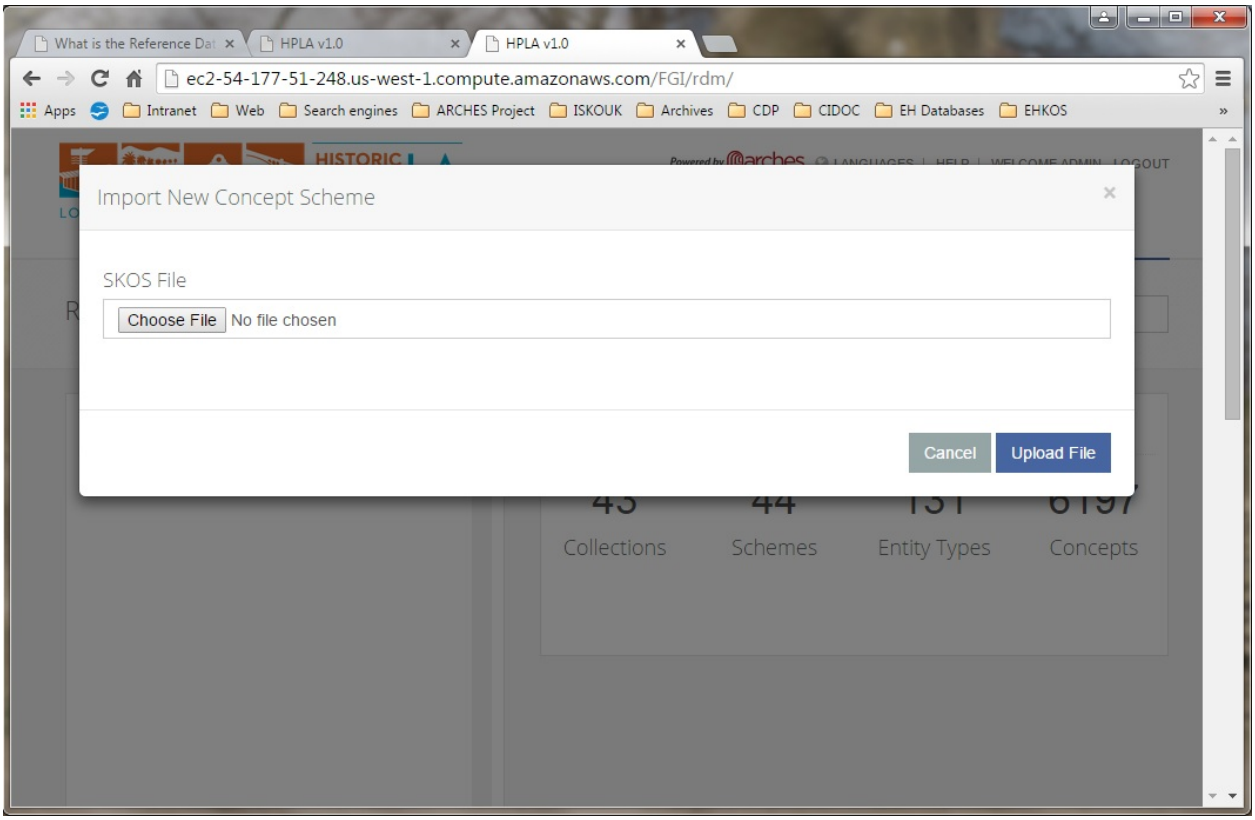
Method 2: Graph view

1. Identify the concept's parent concept and bring up its details.
2. In the *Broader/Narrower Concepts* panel make sure the graph view is visible by clicking on **Show graph**.
3. Click on the node for the concept to be deleted. A dialog box with the concept label and a 'x' symbol will appear.
4. Click on the '**x**' symbol next to the concept to be deleted. The *Delete a Concept* pop-up will appear.

   NB A warning message 'By deleting this concept, you will also be deleting the following concepts as well. This operation cannot be undone.' will appear. If you do not want to delete the concept click the **No** button.
5. Click the **Yes** button. The concept is deleted (along with any of its children) and the node will disappear.

Importing a scheme

1. In the left hand panel select **Import Scheme** from the **Tools** dropdown. The *Import New Concept Scheme* pop-up will appear.
2. Click the **Choose File** button. The *Windows Explorer* panel will appear.
3. Navigate to the file to be uploaded. **NB** this should be a SKOS file with a .rdf file type.
4. Click **Open**. You will be returned to the Import New Concept Scheme pop-up and the name of the file will have populated the form.
5. Click **Upload File**. The number of Concept Schemes will have increased by 1 and the imported concept scheme will appear in the **ConceptSchemes** panel.



Exporting a scheme

1. In the left hand panel select **Export Scheme** from the **Tools** dropdown. The *Export Scheme* pop-up will appear.
2. Click in the **Select Scheme to Export** field. The *Concept Scheme* dropdown menu will appear.
3. Select the scheme to be exported and click on it. The scheme name will populate the field.
4. Click the **Export** button. A new browser tab will appear containing a SKOS-compliane XML export of the scheme.
5. Right click on the file and select **Save as...** from the pop-up menu. The *Save as* panel will appear.
6. Choose where you want to save the file by navigating through the folder sturcture.
7. Give the file a relevant name and click the **Save** button. The file will be saved to the selected location.

Deleting a scheme

1. In the left hand panel select **Delete Scheme** from the **Tools** dropdown. The *Delete Scheme* pop-up will appear.

   NB A warning message stating 'You won't be able to undo this operation! Are you sure you want to permanently delete this entire scheme from Arches?' will appear. If you do not want to delete the scheme click the **Close** button.
2. Click in the **Select Scheme to Delete** field. The *Concept Scheme* dropdown menu will appear.
3. Select the scheme to be deleted and click on it. The scheme name will populate the field.
4. **Click the Delete button. NB** The warning message will appear again along with a list of all of the concepts to be deleted. If you do not want to delete the scheme click the **Close** button.

5. Click the **Delete** button to confirm deletion. The scheme is deleted along with all its concepts.

CHAPTER 11

Packages

A package is an external collection of arches data (resource models, business data, concepts, collections) and customization files (widgets, datatypes, functions, system settings) that you can load into an arches project.

Loading a Package

1. To load a package simply run the load_package command using your *project's manage.py file:

```
python manage.py packages -o load_package -s https://github.com/package/
↪archive/branch.zip -db true
```

-db	*true* to run setup_db to rebuild your database. default = 'false'
-ow	*overwrite* to overwrite concepts and collections. default = 'ignore'
-st	*stage* to stage concepts and collections. default = 'stage'
-s	a path to a zipfile located on github or locally
-o	operation name

If you do not pass the *-db True* to the load_package command, your database will not be recreated. If you already have resource models and branches with the same id as those you are importing, you will be prompted to confirm whether you would like to keep or overwrite each model or branch.

*It is important to note that you cannot load a package directly into core Arches. Currently packages must be loaded into a project. *Create a project*.

If you are a developer running the latest arches you probably want to create a project with a new arches installation. This ensures that the *arches_project create* command uses the latest project templates.

(a) Uninstall arches from your virtualenv

```
pip uninstall arches
```

(b) Navigate into arches root folder delete the *build* directory

(c) Reinstall arches

```
python setup.py install

python setup.py develop
```

(d) Navigate to where you want to create your new project and run:

```
arches-project create mynewproject
```

(e) Finally run the *load_package* command using the projects manage.py file.

```
python manage.py packages -o load_package -s https://github.com/
↪package/archive/branch.zip -db true
```

Creating a New Package

If you want to create additional projects with the same data or share your data with others that need to create similar projects, you probably want to create a package.

The *create_package* command will help you get started by generating the folder structure of a new package.

1. To create new package simply run the create_package command. The following example would create a package called *mypackge*.

```
python manage.py packages -o create_package -d /Full/path/to/mypackage
```

-d full path to the package directory you would like to create

-o operation name

2. Below is a list of directories created by the *create_package* command and a brief description of what belongs in each. Be sure not to place files that you do not want loaded into these directories. If, for example, you have draft business_data that is not ready for loading, just add a new directory and stage your files there. Directories other than what is listed below will be ignored by the loader.

business_data Resource instance .csv and corresponding .mapping files, each sharing the same base name.

business_data/files Files to be added to the uploaded files directory

business_data/relations Resource relationship files (.relations)

extensions/function Each function in this directory should have its own directory with a template (.htm), viewmodel (.js) and module (.py). Each file must share the same base name.

extensions/datatypes Each datatype in this directory should have its own directory with a template (.htm), viewmodel (.js) and module (.py). Each file must share the same base name.

extensions/widgets Each widget in this directory should have its own folder with a template (.htm), viewmodel (.js) and configuration file (.json). Each file must share the same base name.

graphs/branches arches.json files representing branches

graphs/resource_models arches.json files representing resource models

map_layers/mapbox_styles/overlays* Each overlay should have a directory with a mapbox style as exported from mapbox including a *style.json* file, *license.txt* file and an *icons* directory

map_layers/mapbox_styles/basemaps* Each basemap should have a directory with a mapbox style as exported from mapbox including a *style.json* file, *license.txt* file and an *icons* directory

map_layers/tile_server/overlays* Each overlay should have a directory with a *.vrt* file and *.xml* to style and configure the layer. Each file must share the same base name.

map_layers/tile_server/basemaps* Each overlay should have a directory with a *.vrt* file and *.xml* to style and configure the layer. Each file must share the same base name.

reference_data/concepts SKOS concepts .xml files

reference_data/collections SKOS collection .xml files

system_settings The system settings file for your project

*** map layer configuration** By default mapbox-style layers will be loaded with the name property found in the layer's style.json file. The default name for tile server layers will be the basename of the layer's xml file. For both mapbox-style and tile server layers the default icon-class will be *fa fa-globe*. To customize the name and icon-class, simply add a meta.json file to the layer's directory with the following object:

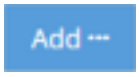
```
{
    "name": "example name",
    "icon": "fa example-class"
}
```

3. It is not necessary to populate every directory with data. Only add those files that you would like to share.

   Once you've added the necessary files to your package, simply compress it as a zip file or push it to a github repository and it's ready to be loaded.

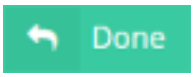
CHAPTER 12

Create a New Resource Model

1. From the Arches Designer front page, click 

and choose "New Resource Model". You are now presented with the Settings for your new Resource Model. (If you choose 

in the top left of the view, you will see other available Arches Designer tools, all of which we'll use later.)
2. In the Identifiers tab, set the Name to "Building".
3. Set Ontology to "CIDOC CRM v6.2". (To learn more about what this means, read *Ontologies in Arches*.)
4. In the Root Class dropdown, type "18" and choose "E18_Physical_Thing".
5. In the Icon tab, type "building" to find the building icon, and select it.
6. We can skip over the Description tab for now, as the data stored there is not necessary at this point.
7. Finally, in the Status tab, make sure your model is Inactive. You don't want someone using the Building model before you are done creating it!
8. Click 

to save your edits.
9. Click 

to return to the Arches Designer home page.

This is all you need to do to create the new Resource Model, but it's not ready for use yet. We still need to add at least one Branch to it. Read *Create a Branch* to make a new Branch from scratch, or skip to *Add a Branch to a Resource Model* if you already have a Branch ready to go.

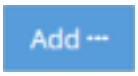
CHAPTER 13

Create a Branch

Now that you've created a new Resource Model to record buildings, you need to add some Branches to it. A Branch is a configuration for a subset of information that you will store about the building. We're going to make a branch to record the Name. This is done by completing the following tasks.

Configure the Branch Settings

Just as you created the Resource Model by defining a few settings, this will be your first step toward creating a Branch.

1. From the Arches Designer home page, choose 

> New Branch
2. You are now presented with the Settings menu for your new Branch. You will notice only small differences between these settings and those available for the Resource Model, but if you click 

, you will see that a Branch has fewer designer tools than a Resource Model.
3. In the Identifiers tab, add "Name" as the Name, and choose "CIDOC CRM v6.2" for the ontology. Now, type "41" in the Root Class dropdown, and choose "E41_Appellation".
4. In the Icon tab, type "tag" and choose the tag icon.
5. Again, you can skip the Appearance and Description tabs, unless you want to add extra information.
6. Finally, in the Status tab, set your Branch to Active.
7. Click 

.

Create a Basic Branch Graph

Now that you've defined the Branch settings, you need to create its graph. A graph is a collection of nodes, where each node defines a piece of information that will be collected to describe your resource. In this Branch we will make a very simple graph, with a node to collect the name of the building. See About the Graph Designer for more information.

1. From Manage ··· choose Graph/Semantics to enter the Graph Manager.
2. First, change the Node Name of your Top Node from “Top Node” to “Name”.
3. Now, notice that the CRM Class is “E41_Appelation”. This is because we defined that class as our Branch’s root class in the settings.
4. Next, under Data Type, change the selection to “string”, and click Save Edits
5. Now that our top node looks good, we can add a second node to the graph. Click ⊕ to begin this process.
6. You now see a list of the nodes you can add. Select the one called Node and click Append to current node
7. In your new node, change the Node Name to “Name Type”.
8. In CRM Class, type “55” to find “E55 Type”.
9. In Relationship to Name choose “P2_has_type”.
10. Under Data Type, select “concept”.
11. In Concept Collection, choose “Name Type”.
12. Click Save Edits.

You have now completed your first graph, and it should look like this:

Configure the Branch’s Card

By creating the graph, you defined what information will be contained in this Branch. In our example so far, the branch will store a Name value, linked to a Name Type value. However, you still need to define how the user will enter that information. This is done through Widgets, which are likewise stored in Cards.

1. From Manage ··· choose Cards.
2. Once in the Card Manager, you are presented with a default configuration, based on the contents of the graph. Because we have a very simple graph, the card is very simple also.
3. Change the Card Tab Title to “Identification”, and add instructions if desired.
4. You may find that the Name Type dropdown widget is positioned above the Name text input widget. You can change the order of these widgets by dragging Name Type below Name in either the Card contents pane on the left, or in the Card mockup itself on the right.
5. Click Done.

You have now finished creating a new Branch, albeit, a very simple one. First, you updated the settings–name, ontology, etc.–of the Branch. Then, you created the graph that models what information will be collected through this Branch, and, finally, you configured the Card that will be used to collect this information. Now, we have to add this

Branch to the Resource Model we created earlier. Return to the Arches Designer main page and head to *Add a Branch to a Resource Model*.

CHAPTER 14

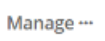
Add a Branch to a Resource Model

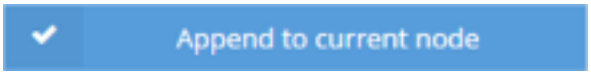
Now that you have created a Resource Model (Building), and a new Branch (Name), the next step is to add the Branch to the Resource Model. From the Arches Designer main page, click Find and use the dropdown to locate your model called Building. Click on this model to begin editing it; you should come to the Settings menu.

There are actually two methods for adding a branch to a Resource Model:

1. By appending the Branch's graph of the Resource Model's graph
2. By adding the Branch's Card to the set of Cards used by the Resource Model

Method 1: Append the Branch to a Resource Model's Graph

1. From 

choose Graph/Semantics to enter the Graph Designer.
2. You will see the graph for your Resource Model, with a Top Node that represents the Resource itself. While we're here, let's rename this node "Building".
3. Next, we'll add the graph from the Name Branch (created in *Create a Branch*). Click  in the node manager pane.
4. You will be presented with a list of all the Branches that can be appended to your current top node. (Note that because we have chosen for this Resource Model to use the CIDOC CRM ontology, you are only seeing Branches listed whose own CRM Root Class can legally be attached to the current top node).
5. Choose the Name Branch, and click 

6. Click on the new Name and Name Type nodes. Note that the CRM Class and Relationship from Name/Building are the same as those that you set earlier when you created that Branch.
7. Click 

.

You can now go to the Data Widgets and Cards menu, and you'll find the Card for this Branch has automatically been added.

Method 2: Add the Branch's Card to the Resource Model

1. From  choose Cards to enter the Card Designer.

2. Click "Add Card" to open the Card Library. You are now looking at list of the Cards from all the Branches that are available to you. Find the Name Card (created in *Create a Branch*) and drag it into main panel. Note: If you don't see the Name Card, it may still be set as "inactive".

That's all! You can now head to the Graph Designer to see that a new branch has been added to the graph.

CHAPTER 15

Create a Menu

Now that you've adding at least one Card to the Resource Model, you are ready to put together the first Menu. A Menu is a collection of Cards, which allows you to group the data entry process into themed sections.

1. From Manage ··· choose Forms to enter the Form Designer.
2. Click "Add Menu"
3. On the left hand, you'll see the Data Card Library, where all of the Cards available to you are listed. The central area will hold the Cards that you have selected for your form, and on the right are your new Menu's settings. We'll begin by defining these settings.
4. In the Identifiers tab, set the Menu Name to "Identification".
5. In the Icon tab, type "tag" to find the tag icon, and select it.
6. In the Status tab, make sure that the form is Visible.
7. Click 

to save your progress.
8. Now you just have to add the Card to the Menu. In the Data Card Library (left pane), you should see the Name Card. Drag it into the central area. If we had created more Branches, we'd be able to add their Cards to this Menu as well.
9. Click 

to finish your work on this Menu.

Even though we only have one Card at this point, and a very simple Identification Menu, you should now see how easy it is to create and configure Menus. Note that if you have more than one Card in your Menu, the order you set them in the central area will be the order they appear in the Menu.

CHAPTER 16

Add a Function to the Resource Model

Functions allow Arches to automatically perform operations on input data after it is entered by users. A simple example that comes built-in with Arches is the function called "Define Resource Descriptors". It allows you to define how a resource will be labeled throughout the app–in search results, in map popups, etc.

1. From Manage ··· choose Functions to enter the Function Manager.
2. Select the "Define Resource Descriptors" Function. You'll see 3 tabs, each used to define a resource descriptor that appears throughout the Arches interface. For our purposes, we'll only deal with the Display Name tab.
3. Make sure the "Name" card is selected in the Card Name dropdown. This will pre-populate the Primary Name Template line with all of the nodes in the "Name" card.
4. Change the template from `<Name>, <Name Type>` to `<Name> (<Name Type> Name)`. With this template, a building with a primary name of "Scott House" will have a display name of "Scott House (Primary Name)".
5. Click Done.

CHAPTER 17

Create a Report

The final step in configuring our Resource Model is to create a report. Reports aggregate and display information that has been entered for a resource, and also allow you to keep certain information hidden from the public. You can also define multiple reports for a given Resource Model, which allow for even more flexibility in how you publish your data.

1. From Manage ··· choose Reports to enter the Report Designer.
2. Click “Add Report”.
3. You now see the list of report templates that you must choose from. Because we have very simple information to display (just a name), we’ll choose the most basic “No Header Template”.
4. Drag the “No Header Template” onto the right side of the screen.
5. Now click on your report, which will be named “New Report” by default.
6. On the left, you’ll see the Report Manager, which shows all of the sections in the report.
7. Select the Header section, click the Active button, and type “Building Resource” for the Report Title.
8. Click Back to Reports.

CHAPTER 18

Arches HIP Contents

By default, Arches ships with a set of cohesive Resource Models, Branches, and Concepts which form the v4 version of Arches-HIP (Heritage Inventory Package), a data schema that was originally developed for Historic Places L.A.. HPLA is the Arches implementation in Los Angeles that was designed to hold data from the ongoing Survey LA project.

To import this entire data schema into your Arches database, use these commands (with your virtual environment activated):

```
python manage.py packages -o import_graphs
python manage.py packages -o python manage.py packages -o import_reference_data -s␣
↪arches/db/schemes/arches_concept_scheme.rdf -ow overwrite -st keep
python manage.py packages -o python manage.py packages -o import_reference_data -s␣
↪arches/db/schemes/arches_concept_collections.rdf -ow overwrite -st keep
```

The package comprises 6 Resource Models and 36 Branches.

Resource Models

- **Activity Resource Model** This resource model describes activities relating to heritage resources and heritage resource groups.
- **Actor Resource Model** This resource model describes actor resources such as individual people and groups of people.
- **Heritage Resource Group Resource Model** This resource model describes heritage resource groups which are groupings of historically significant resources. Those historically significant resource may more may not themselves be instances of Heritage Resource.
- **Heritage Resource Model** This resource model describes heritage resources, which includes monuments, buildings, structures, etc.
- **Historical Event Resource Model** Resource Model for the Historical Event Resource (E5), which is used to describe significant historical events.

- **Information Resource Model** This resource model defines information resources, such as images, reports, and publications.

Branches

- **Activity Phase** Describes the time span and type of an Activity Resource.
- **Actor Phase** Phase Type Assignment for the Actor resource model.

  Connects to Actor E39 via P41i
- **Appellation** Describes an appellation assigned to an Actor Resource.

  Relates to resource model via P131
- **Beginning of Existence** Describes the type and time span of the beginning of a resource's existence.

  Relates to the Actor resource model E39 via P92i. Relates to Historic Event E5 and Activity E7 via p116i.
- **Component** Physical thing on a heritage resource. Connect to Heritage Resource E18 via P46
- **Condition Assessment** Describes the conditions, threats, and disturbances affecting a Heritage Resource or Heritage Resource Group. Additional information may include a management recommendation, a condition image, the data the condition was assessed and a description of the condition.
- **Description** Generic branch for capturing free-form written descriptive information. Intended to be evolution from v3 description branch from HIP 3. Created semantic description node with description string and description type hanging off of it. Description (semantic node) becomes a grouping node (type and string).

  This branch meets technical business rules for CRM compliance but CRM experts have said that this is an inappropriate implementation of E62.

  Connects to all resources with P140i
- **End of Existence** Describes the type and time span of the end of a resource's existence.

  Relates to the Actor E39 Resource Model via P93i
- **Evaluation** This branch evaluates instances of Heritage Resource E18 and Heritage Resource Group E27, and is based on the City of Los Angeles' SurveyLA survey methodology for evaluation of significance and eligibility for designation as a heritage resource or heritage resource group. Evaluation consists of defining a historic context (Evaluation Criteria) which triggers specific eligibility requirements (Eligibility Requirements) and results in the selection of one or more status codes (Status) and the writing of a reasons statement (Reasons). The branch also contains nodes to record the date of Evaluation, the dates defining the Period of Significance for evaluation.

  Relates to E18 and E27 resource models via P140i.
- **Event Phase Type** Describes a Historical Event Type within a given timespan/phase.

  Connects to the Historical Event resource model via P10
- **Existence** Start/End dates for resources. Assumed to be typed and within calendar time.

  Connects to Activity (E7) and Historical Event (E5) resource models via P114 Connects to Heritage Resource (E18) and Heritage Resource Group (E27) resource models via P12i Connects to Actor (E39) resource model via P11i
- **External Identifier** Used to hold identifiers necessary to link a given resource to records in an external (perhaps legacy) system.

- **Heritage Resource Group Phase Type** Phase Type Assignments for Heritage Resource Group E27

  Connects via P92i

- **Heritage Resource Phase Type** Phase Type Assignments for Heritage Resource E18.

  Connects via P92i

- **Information Carrier** An object or substance used to record and accumulate data
- **Information Resource Copyright** Used to define legal privileges associated with an Information Resource.
- **Information Resource Creation Event** Describes the creators, contributors, type, and date of a creation and/or update event of an information resource.
- **Keyword** An informative word used for resource information retrieval that indicates the content of a resource.

  Relates to resources via P2.

- **Language** This branch describes the language of an Information Resource.
- **Measurement** The Measurement branch measures instances of a Heritage Resource E18 or a Heritage Resource Group E27

  Relates to those resource models via P39i

- **Modification Event** Describes the modification of a Heritage Resource.

  Connects to Heritage Resource (E18) via P12i

- **Name** The name of a resource.

  Applies to Heritage Resource, Heritage Resource Group, Activity, and Historical Event. Actors receive their "names" from the Appellation branch.

- **Place 1** Describes the physical location of a heritage resource or heritage resource group. Includes extra nodes for cadastral information.

  Relates to Heritage Resource E18 with P53 Relates to Heritage Resource Group E27 with P89

- **Place 2** For Actor resources: Describes the former and current residences of an Actor resource. Relates to Actor E39 with P74 For Activity and Historic Event: Describes the location of where an activity or historic event took place. Relates to Activity E7 with P7. Relates to Historical Event E5 with P7
- **Place 3** Describes the location of an information resource. Relates to Information Resource E73 with P67
- **Publication Event** This branch describes the publication of an Information Resource.

  Connects to Information Resource (E73) with -P128

- **Resource Type Classification** Describes the type of resource.

  Applies to: Heritage Group, Heritage

- **Resource Update Event** This is intended as a sub-branch to Resource Creation Event within the Information Resource Resource Model. Stores the date when an update to the information resource occurred.
- **Right** Captures information about special status, protection, or privileges afforded under law.

  Applies to: Heritage Resource, Heritage Resource Model

- **Settings Basic Search** Configuration settings for search returns
- **Settings Saved Searches** System configuration for saving searches
- **Settings Search Map** Initial settings for Arches map in the Search UI
- Settings System Defaults

- **Settings Time Search** Configuration Settings for temporal search
- **Temporal Coverage** Defines time span for which the Information Resource is relevant. Applies to: Information Resource
- **Title** Describes the title of an Information Resource.

CHAPTER 19

Glossary

Glossary of Arches v4 Terminology

- **Arches Designer** - A user interface for facilitating database design, i.e. the creation of Resource Models. The Arches Designer consists of many different tools, such as the Graph Designer and Card Manager, each of which helps build a different facet of Resource Model creation.
- **Basemaps** - Underlying map layers which include, by default, aerial imagery, streetmaps, or terrain. You can also add your own basemaps through the same process as adding overlays.
- **Branch** - Branches are building blocks that aid in the creation of resource models. When you add a branch to a resource model, its contents are copied into the resource model. This allows you to further customize the resource model while leaving the original branch unaltered.
- **Card** - Cards are used to configure the data entry representation of a branch's graph; they define how information will be collected for each nodegroup. In some cases a complex branch may have multiple cards, whcih will be aggregated into a card container. Cards contain widgets, and determined how the widgets are ordered in the user interface.
- **Concept** - A vocabulary term that is used throughout the Arches database to define resource. A concept has a preferred label ("house") and may have any number of alternative labels ("domicile", "townhouse"). When searching your database, a search for "domicile" would automatically use the "house" concept.
- **Concept Collections** - Concepts are grouped into collections. An example would be the concepts "Eastlake", "Italianate", and "Queen Anne", all of which would be grouped in an "Architectural Style" concept collection.
- **Datatype** - A defined type of business data, such as a number or a date. Each node has a datatype.
- **Resource Model** - Resource Models are top-level categories for resources in your database. When creating new resources, a data entry user must decide which resource model to use, thereby defining what information is collected for the resource. The entire Arches Designer exists to facilitate the creation and customization of resource models.
- **Graph** - A network of nodes, connected by edges, that defines the set of attributes for either a Branch or a Resource Model. If an ontology is enforced on the graph, each node will belong to an ontological class and only certain types of edges may be used to connect them.

- **Menu** - Menus are groupings of cards associated with a given resource model. They allow for an organized, thematic approach to data entry.
- **Node** - The smallest unit of a graph, a node will have a name and datatype. If the graph participates in an ontology, the node must also have a CRM class, and a defined relationship (edge) between it and the node upstream of it.
- **Nodegroup** - Within graphs, nodes are aggregated into nodegroups. An example of a nodegroup would be Name and Name Type. Edit permissions are enforced at the nodegroup level.
- **Ontology** - A set of rules the govern the way nodes are defined and connected in a graph. Arches comes pre-loaded with the CIDOC CRM, an ontology developed by ICOM to model cultural heritage.
- **Overlays** - Static map layers that can be added to Arches. These could be historic maps, administrative boundaries, or existing map services published elsewhere.
- **Reference Data Manager (RDM)** - User interface for managing all of the concepts in your Arches database.
- **Resource** - An entry in your Arches database. Each resource is created using a resource model.
- **Resource Layers** - Map layers that are created from your Arches database. There is one resource layer for each node with datatype 'geojson-feature-collection' that is stored across all resource models.
- **Resource Relationships** - Arches provides the ability to relate one resource to another by creating resource relationships. Resource relationships are directional and will have an associated concept, such as "contains / is contained in".
- **Resource Report** - A resource's report shows all of the saved information for a resource. Templates for reports are associated with each resource models.
- **Time Wheel** - A graphical interface used to support advanced time-based visualization and search of your database.
- **Widget** - An input element designed to manage form input of a specific datatype. Each widget represents one node, and widgets for all nodes in a nodegroup are contained in a single card.