
Arches Documentation

Release 7.5.0

Legion GIS, Farallon Geographics, Coherit Associates

Apr 12, 2024

CONTENTS

- 1 Welcome to the Arches official documentation site! 3**
 - 1.1 Table of Contents: Documentation Topics 3
- HTTP Routing Table 287**

Arches is an open-source data management platform originally developed for the cultural heritage field by the [Getty Conservation Institute](#) and [World Monuments Fund](#). Due to the complex and varied nature of cultural heritage data, and to promote interoperability and sustainable data practices, the Arches Platform has been developed as a standards-based, comprehensive and flexible platform that supports a wide array of uses.

WELCOME TO THE ARCHES OFFICIAL DOCUMENTATION SITE!

This documentation primarily aims to provide guidance with Arches installation, technical administration, management, localization, customization and other extensions. Because Arches sees continual improvement, please help make this documentation provides clear, accurate, and up-to-date information by filing tickets that identify issues for improvement [here on GitHub](#).

1.1 Table of Contents: Documentation Topics

The documentation is organized into the following sections. It is recommended to start with the *Getting Started and Installation* section if you are new to Arches.

1.1.1 Getting Started and Installation

This section provides an introduction to Arches and instructions for installing Arches on your local machine or server.

Introduction

Arches is an enterprise-level system developed to improve data management in support of effective heritage conservation and management. Because Arches has grown in power and flexibility it serves a wide range of needs in the cultural heritage sector and beyond. This section provides an overview of the Arches software release process and different versions of Arches.

Overview

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.

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

- **Geospatial data management** and geoprocessing like a GIS (Geographic 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: <https://community.archesproject.org/>

Version History and Roadmap

The Arches project uses [semantic versioning](#) to describe unique states of the software. Arches was initially released in October 2013 as version 1.0. Since then, Arches has had 7 major releases and many more minor releases and patch releases (see [Arches Release Process](#)). For more details about the capabilities introduced in past versions and capabilities planned for future versions, please review: <https://www.archesproject.org/roadmap/>

Important: License: Arches is free software and is licensed under the terms of the GNU Affero General Public License (<http://www.gnu.org/licenses/agpl-3.0.html>).

Who is Arches for?

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

Documentation Overview

This is the official documentation 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.

Improve Our Documentation! If you find errors, have suggestions, or want to make a contribution, these docs are managed in the [archesproject/arches-docs](#) repo.

Contributing To Arches

Arches is open source software, which means that with your help it will continue to evolve and improve.

- **Bug Reports and Code Contribution** If you find issues with the Arches interface or code, or have the means to contribute code to fix existing issues, please begin by reading our guidelines for [Contributing to Arches](#).
- **Translations** We are always hoping to bring Arches to new audiences around the world. Please post on the [Arches Forum](#) if you are interested in contributing a translation.

Arches Release Process

Starting with version 4.1.0, the Arches team began making available both feature (minor) and patch (micro) releases on a regular basis.

Feature Releases

Feature releases will introduce significant, new features to Arches and will be announced approximately every 6 months. Feature releases may contain schema or API changes that may not be compatible with the previous feature release. Each feature release will be incremented with the pattern *a.b*, where *a* represents the major release and *b* represents the feature (aka minor) release. Each feature release will be placed in its own branch in git, named with its release number followed by an *x* representing the latest patch release. (e.g. `stable/a.b.x`).

Patch Releases

Following each feature release we will resolve bugs, performance, and security issues in the most recent feature release with patch releases. A new patch release, if needed, will be announced every 1 to 3 months and will **not** include breaking changes with the previous patch release. Therefore, we encourage users to stay up-to-date with these releases. Patch releases will be incremented as such: *a.a.b*, *a.a.c*... with *a* representing the feature release and *b* and *c* representing patch (aka micro) releases. In Git each patch release will identified in its feature release branch with a tag.

Release Support

We will release patches only for the latest feature release.

Arches Releases

You can also view [release tags](#) on Github.

Current Release

- [7.5.0 - notes](#)

Past Releases

- [7.4.3 - notes](#)
- [7.4.2 - notes](#)
- [7.4.1 - notes](#)
- [7.4.0 - notes](#)
- [7.3.0 - notes](#)
- [7.2.1 - notes](#)
- [7.2.0 - notes](#)
- [7.1.1 - notes](#)
- [7.1.0 - notes](#)
- [7.0.0 - notes](#)
- [6.2.6 - notes](#)
- [6.2.5 - notes](#)
- [6.2.4 - notes](#)
- [6.2.3 - notes](#)
- [6.2.2 - notes](#)
- [6.2.1 - notes](#)
- [6.2.0 - notes](#)
- [6.1.2 - notes](#)
- [6.1.0 - notes](#)
- [6.0.1 - notes](#)
- [6.0.0 - notes](#)
- [5.1.4 - notes](#)
- [5.1.3 - notes](#)
- [5.1.1 - notes](#)
- [5.1.0 - notes](#)
- [5.0.0 - notes](#)
- [4.5.0 - notes](#)
- [4.4.1 - notes](#)
- [4.3.1 - notes](#)
- [4.3.0 - notes](#)
- [4.2.0 - notes](#)
- [4.1.1 - notes](#)
- [4.1.0 - notes](#)
- [4.0.1 - notes](#)
- [3.1.2 - notes](#)

- [3.1.1 - notes](#)
- [3.0.4 - notes](#)
- [3.0.3 - notes](#)
- [3.0.2 - notes](#)
- [3.0.1 - notes](#)
- [3.0 - notes](#)

Installing

This section of the documentation provides guidance on how to install Arches. As an “enterprise-level” system, Arches is designed for deployment in organizational contexts with both needs and capabilities beyond those typical of an individual person. While Arches can be installed and tested on a personal computer, it is designed for deployment on servers in a networked environment.

Arches can be deployed for testing on a personal computer provided one has administrative permissions, some comfort and familiarity with command line interfaces, and (typically) some patience with trouble shooting. “Production” deployments (either public or private to an organizational setting) requires some experience with Web hosting, IT systems administration and, if using a cloud service provider, cloud computing infrastructure. These skills are needed to install, configure, and (critically) maintain Arches.

Requirements/Dependencies

System Requirements

Arches works on Linux, Windows, or macOS. Most production implementations use Linux servers.

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

Disk Space

- **2GB** for all dependencies and Arches.
- **8GB** to store uploaded files, database backups, etc.
- Depending on how many uploaded files (images, 3d models, etc) you will have, you may need **much** more disk space. We advise an early evaluation of how much space you *think* you’ll need, and then provision twice as much just to be safe...

Memory (RAM)

- **4GB**
- This recommendation is based on the fact that Elasticsearch requires 2GB to run, and as per [official Elasticsearch documentation](#) no more than half of your system’s memory should be dedicated to Elasticsearch.
- In production, you very likely need to increase your memory. In building the production (mini-fied) frontend asset bundle, yarn (all by itself!) will require at least 8GB to run. If you don’t have enough memory, yarn will likely return an error, sometimes after several minutes or hours of processing. In production, you may also find it useful to allow Elasticsearch to use [up to 32GB](#).

Software Dependencies

Arches requires the following software packages to be installed and available. Ubuntu Linux users see below for an installation script.

Python >= 3.10

- Installation: <https://www.python.org/downloads/>
- Python 3.10 and later comes with pip
- **Windows** You must choose 32-bit or 64-bit Python based on your system architecture.
- **macOS** This guide works well if you wish to install via *brew*: <https://docs.python-guide.org/starting/install3/osx/>

Git >= 2.0

- Installation: <https://git-scm.com/downloads>
- **Windows** Choose the “Use Git from the Windows Command Prompt” option during installation.
- **macOS** You can install Git via *brew*: <https://brew.sh/>

PostgreSQL >= 12 with PostGIS 3

- **macOS** Use [Postgres.app](#).
- **Windows** Use the [EnterpriseDB installers](#), and use Stack Builder (included) to get PostGIS. After installation, add the following to your system’s PATH environment variable: C:\Program Files\PostgreSQL\12\bin. Make sure you write down the password that you assign to the postgres user.

Elasticsearch 8

- Installers: <https://www.elastic.co/downloads/past-releases/elasticsearch-8-5-1>
- Elasticsearch is integral to Arches and can be installed and configured many ways. For more information, see [Arches and Elasticsearch](#).

GDAL >= 2.2.x

- **Windows** Use the [OSGeo4W installer](#), and choose to install the GDAL package (you don’t need QGIS or GRASS). After installation, add C:\OSGeo4W64\bin to your system’s PATH environment variable.

Node.js 16.x (recommended)

- Installation: <https://nodejs.org/> (choose the installer appropriate to your operating system).
- NOTE: Arches may not be compatible with later versions of Node.js (after 16) (see [discussion](#)).

Yarn >= 1.22, < 2

- Recommended Installation: <https://classic.yarnpkg.com/en/docs/install> (One can also install Yarn via *apt* on Linux operating systems, see [example](#)).
- NOTE: We are pointing to the “classic” yarn installer to avoid installation of more recent versions of yarn that are not compatible with Arches via the Node.js [package manager](#).

To support long-running task management, like large user downloads, you must install a Celery broker like RabbitMQ or Redis:

Brokers

- Options: <https://docs.celeryproject.org/en/stable/getting-started/first-steps-with-celery.html#choosing-a-broker>

- Once you have a broker installed, read more about [Task Management](#) in Arches.

Scripted Dependency Installation

For Ubuntu we maintain an [ubuntu_setup.sh](#) script to install dependencies. It works for 18.04 and 20.04, and preliminary testing shows it to be compatible with 22.04 as well.

```
wget https://raw.githubusercontent.com/archesproject/arches/stable/7.5.0/arches/install/
ubuntu_setup.sh
source ./ubuntu_setup.sh
```

You will be prompted before each dependency is installed, or use `yes | source ./ubuntu_setup.sh` to install all components (Postgres/PostGIS, Node/Yarn, and ElasticSearch).

Installing Core Arches

Most of the instructions here will focus on installation of Arches on the Ubuntu distribution of Linux. Arches can also be installed on Windows and macOS, but installation on those operating systems will likely require additional configuration and debugging.

See also:

If you plan to extend or contribute to Arches, please see [Creating a Development Environment](#).

See also:

We have an in-progress [Docker install](#), and would love help improving it. You can also review some works-in-progress and community-created approaches to using Docker [Installation with Docker](#)

Installation on Windows via WSL

Some of the directions below will provide some guidance to install Arches dependencies and core Arches on machines running the Windows operating system. However, installation on Windows will likely require more configuration (and troubleshooting) than installation on a Linux distribution like Ubuntu. Fortunately, the Windows operating system has a feature called “[Windows Subsystem for Linux](#)” (WLS) that allows one to run a Linux environment on a Windows machine. With WSL, one can install Arches dependencies and core Arches on an Ubuntu (or other Linux distribution) virtual machine. The steps for installing Arches on a WSL Ubuntu virtual machine will be identical to the steps used to install Arches on an “bare-metal” Ubuntu machine.

Currently, WSL comes in two architectures. We recommend using the current default “WSL 2” version of WSL because it has a better file system performance and other benefits. It should be simpler and easier to install Arches on Windows machines via WSL (especially WSL 2).

Create a Virtual Environment

Virtual Environment Reference

If you are unfamiliar with virtual environments, please take a look at the [Python documentation](#) before continuing.

Regardless of the your machine's operating system, you'll need to have Python installed and the capability to start Python virtual environments (see [introduction](#)). For Arches, you'll need a **Python 3.10+** virtual environment. *Skip ahead* if you have already created and activated one. Otherwise, use the commands below for a quick start.

Create a virtual environment:

```
python3 -m venv ENV
```

This will generate a new directory called `ENV`.

Note: On some linux distributions, if the python version is less than 3.8, entering the following command may yield an error but it should alert you to any dependencies you may need to install, after which you'll be able to run this command.

Activate the virtual environment

The following are relative paths to an `activate` script within `ENV`.

Linux and macOS:

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

Windows:

```
ENV\Scripts\activate.bat
```

Note: After you activate your virtual environment, your command prompt will be prefixed with `(ENV)`. From here on the documentation will assume you have your virtual environment activated. Run `deactivate` if you need to deactivate the virtual environment.

Test the Python version in ENV:

```
python
```

This will run the Python interpreter and tell you what version is in use. If you don't see at least 3.8, check your original Python installation, delete the entire `ENV` directory, and create a new virtual environment. Use `exit()` or `ctrl+C` to leave the interpreter.

Upgrade pip

A recommended step, though not always strictly necessary:

```
python -m pip install --upgrade pip
```

Install Arches with pip

Use the following to get the latest stable release of Arches:

```
pip install arches
```

Common Errors

Creating a New Arches Project

A Project holds branding and customizations that make one installation of Arches different from the next. The name of your project must be **lowercase** and use **underscores** instead of spaces or hyphens. The example below uses `my_project`.

Create a Project

Linux and macOS:

```
arches-project create my_project
```

Windows:

```
python ENV\Scripts\arches-project create my_project
```

Common Errors

Note: You can add `--directory path/to/dir` to change the directory your new project will be created in.

Warning: On Windows, open `my_project\my_project\settings_local.py` and add the following line:

```
GDAL_LIBRARY_PATH = "C:/OSGeo4W64/bin/gdal201.dll"
```

Be sure to adjust the path as necessary for your GDAL installation, and note the *forward* slashes.

Setup the Database

First, enter the project directory:

```
cd my_project
```

and then run:

```
python manage.py setup_db
```

Note: You may be prompted to enter a password for the `postgres` user. Generally, our installation scripts set this password to `postgis`, however you may have set a different password during your own Postgres/PostGIS installation.

Common Errors

Build a Frontend Asset Bundle

In your current terminal, run the Django development server (with the Arches virtual environment activated):

```
python manage.py runserver
```

Then, in a second terminal, activate the virtual environment used by Arches (this is a required step). Then navigate to the root directory of the project. (you should be on the same level as *package.json*) and build a frontend asset bundle:

```
cd my_project/my_project
yarn build_development
```

If you have trouble with this step, see [Troubleshooting Frontend Builds](#) below.

Note: `yarn build_development` creates a static frontend asset bundle. Any changes made to frontend files (eg. `.js`) will not be viewable until the asset bundle is rebuilt. run `yarn build_development` again to update the asset bundle, or run `yarn start` to run an asset bundler server that will detect changes to frontend files and rebuild the bundle appropriately.

View the Project in a Browser

Navigate to `localhost:8000` in a browser. Use `ctrl+C` to stop the server.

Configure the Map Settings

The first thing everyone wants to do is look at the map, so let's set this up first.

1. Go to Mapbox.com and create a free account.
2. Find your default API key (starts with `pk.`) and copy it.
3. Now go to `localhost:8000/settings`.
4. Login with the default credentials: **username:** admin **password:** admin
5. Find the **Default Map Settings**, and enter your *Mapbox API Key* there.
6. Feel free to use the ? in the top-right corner of the page to learn about all of the other settings, and change any that you like (heed warning below).
7. Save the settings.
8. Navigate to `localhost:8000/search` to make sure the basemap appears.

Note: We recommend exporting these settings by running `python manage.py packages -o save_system_settings`. This will create a JSON file in your project, which will be used if you ever need to setup your database again.

Warning: If you create a new **Project Extent**, you should also update the **Search Results Grid** settings, otherwise you could get a JSON error in the search page. To be on the safe side, choose a high *Hexagon Size* combined with a low *Hexagon Grid Precision*.

Load a Package

An Arches “package” is an external container for database definitions (graphs, concept schemes), custom extensions (including functions, widgets, datatypes) and even data (resources). Packages are installed into projects, and can be used to share schema between installations.

To get started, load this sample package:

```
python manage.py packages -o load_package -s https://github.com/archesproject/arches-
example-pkg/archive/master.zip -db
```

Go to `localhost:8000/graph` to see 6 Resource Models that you can now use. You can also create new Resource models from scratch.

Go to `localhost:8000/resource` to begin creating resources based on one of these resource models.

Go to `localhost:8000/search` to find and inspect resources that you have created.

You can add `-dev` to the `load_package` command to create a few test user accounts.

What Next?

- Read more about [Projects and Packages](#)

Common Errors

- On macOS, If you get this error

Error: *ValueError: -enable-zlib requested but zlib not found, aborting.*

try running `xcode-select --install` ([reference](#))

- Getting a connection error like this (in the dev server output or in the browser)

Error: *ConnectionError: ConnectionError(<urllib3.connection.HTTPConnection object at 0x0000000005C6BC50>: Failed to establish a new connection: [Errno 10061] No connection could be made because the target machine actively refused it) caused by: NewConnectionError(<urllib3.connection.HTTPConnection object at 0x0000000005C6BC50>: Failed to establish a new connection: [Errno 10061] No connection could be made because the target machine actively refused it)*

means Arches is not able to communicate with ElasticSearch. Most likely, ElasticSearch is just not running, so just start it up and reload the page. If you can confirm that it *is* running, make sure Arches is pointed to to correct port.

- Postgres password authentication error

Error: *django.db.utils.OperationalError: FATAL: pw authentication failed for user postgres*

Most likely you have not correctly set the database credentials in your `settings.py` file. Many of our install scripts set the db user to `postgres` and password to `postgis`, so that’s what Arches looks

for by default. However, if you have changed these values (particularly if you are on Windows and had to enter a password during the Postgres/PostGIS installation process), the new values must be reflected in `settings.py` or `settings_local.py`.

Note: On Windows, you can avoid having to repeatedly enter the password while running commands in the console by setting the `PGPASSWORD` environment variable: `set PGPASSWORD=<your password>`.

Troubleshooting Frontend Builds

Building the frontend assets can sometimes be a source of challenge and frustration. Sometimes a “locked down” computer (with strict security configurations) may cause some trouble. If this is the case, you can try the following steps to iterate toward a successful build.

1. **Edit your `.yarnrc` file to disable strict SSL.**

To do so, navigate to your project’s root directory and open the `.yarnrc` file in a text editor. Add the following lines to the end of the file: `.. code-block:: bash`

```
cafile null strict-ssl false
```

2. **After the above edits, save the file.**

3. **Remove the `node_modules` folder and `yarn.lock` file if they exist:**

```
cd path/to/dir/my_project/my_project
rm -rf node_modules
rm yarn.lock
```

4. **If you’re using a virtual environment, activate it. `ENV` should be replaced with the name of your virtual environment.**

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

5. **Run your Arches Django server and leave it running.**

```
python manage.py runserver
```

6. **Open a *new terminal* to complete the following steps below.**

7. **If you’re using a virtual environment, activate it as in step 4 above. `ENV` should be replaced with the name of your virtual environment.**

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

8. **Navigate to the same directory as `package.json`, and install the frontend dependencies:**

```
cd path/to/dir/my_project/my_project
yarn install
```

9. **Once the dependencies are installed, build your static asset bundle:**

```
yarn build_development
```

If successful, you should see a message indicating that the build was successful. A successful build should make a message looking something like this:

```
cacheable modules 8.62 MiB (javascript) 3.28 KiB (asset) modules by path ./media/ 6.48
MiB 996 modules modules by path ../../ 2.15 MiB (javascript) 3.28 KiB (asset) mod-
ules by path ../../arches/arches/app/media/ 1.2 MiB (javascript) 3.28 KiB (asset) 264 mod-
ules modules by path ../../arches/arches/app/templates/views/ 970 KiB 90 modules ../../arches-
rdm/arches_rdm/media/js/.gitkeep 1 bytes [built] [code generated] ./media/js/ sync ^./.*$ 207
bytes [optional] [built] [code generated] ../../arches/arches/app/media/js/ sync ^./.*$ 18.9 KiB
[optional] [built] [code generated] ../../ENV/lib/python3.10/site-packages/ sync ^./.*$ /media/js/.*$
160 bytes [optional] [built] [code generated] ../../arches-rdm/arches_rdm/media/js/ sync ^./.*$
160 bytes [optional] [built] [code generated] ../../arches/arches/app/media/js/ sync ^./.*$ /me-
dia/js/.*$ 160 bytes [optional] [built] [code generated] ./media/node_modules/moment/locale/
sync ^./.*$ 3.21 KiB [optional] [built] [code generated] webpack 5.89.0 compiled successfully
in 8545 ms Done in 10.71s.
```

Installation with Docker

Why Use Docker?

Docker is a platform that allows you to package, distribute, and run applications in containers. By using Docker, you can install Arches on any system that supports Docker. This helps insulate you from worries about how the operating system or other specifics of your host system impact dependencies and configurations. Some of the benefits of using Docker include:

1. **Isolation:**

Docker allows developers to isolate applications from the underlying infrastructure, reducing the risk of conflicts and making it easier to manage dependencies.

2. **Scalability:**

Docker makes it easy to scale applications up or down, as containers can be started and stopped quickly and easily.

3. **Portability:**

Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, the developer can be assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

We recommend that you gain some understanding about how Docker works as well as some basic proficiency with using Docker before attempting to deploy Arches using Docker. These tutorials can introduce you to the basics of using Docker: <https://docker-curriculum.com/> or <https://www.howtogeek.com/733522/docker-for-beginners-everything-you-need-to-know/>

This document will guide you through the process of using Docker to install Arches on your system. Even if you do not want to use Docker to deploy Arches, the review of Arches Docker setups can still provide a useful guide to see how various dependencies and configurations fit together.

Important: Arches currently lacks an “official” approach to installation using Docker. The examples that we discuss here are drawn from various works-in-progress and community-created approaches. They should be helpful to get started with Docker and Arches, but they are not fully tested for production deployments.

Prerequisites

Before you begin, ensure that your system meets the following requirements:

Docker:

Docker must be installed on your system. You can download Docker from the official website: <https://www.docker.com/get-started>

Docker Compose:

Docker Compose is a tool for defining and running multi-container Docker applications. It must also be installed on your system. You can download Docker Compose from the official website: <https://docs.docker.com/compose/install/>

Example Docker Code Repositories for Testing

The following lists Docker repositories that set up Docker containers running Arches configured for testing and development, not a production deployment.

1. Arches Dependencies

This repo (<https://github.com/archesproject/arches-dependency-containers>) uses Docker to provision the dependency PostgreSQL, ElasticSearch, and RabbitMQ services required by Arches. NOTE: This repo does not install Arches itself, it just provides an alternate means to install dependency services.

2. Arches for Science

This repo (<https://github.com/archesproject/arches-for-science-prj>) uses Docker to deploy an instance of Arches running the package and extensions for the *Arches for Science* project. This Docker deployment is designed to run in conjunction with the Docker containers and Docker network started by the *Arches Dependency* repo discussed above. The Arches for Science project aims to support workflows in the scientific conservation of objects (especially in museum collections), see: <https://www.archesproject.org/arches-for-science/>

3. Arches HER (Historic England)

This repo (<https://github.com/archesproject/arches-her>) uses Docker to deploy an instance of Arches running the package and extensions for the Arches implementation developed for Historic England (<https://www.archesproject.org/arches-for-hers/>). This Docker deployment is designed to run in conjunction with the Docker containers and Docker network started by the *Arches Dependency* repo discussed above.

The *Arches for Science* Docker repository and the *Arches HER* Docker repository both launch instances of Arches that depend upon Docker containers and the Docker network started by the *Arches Dependencies* repository. These Docker repositories can be used to “spin up” different versions of Arches and Arches dependencies. You need to launch the appropriate set of dependencies started with *Arches Dependencies* with the version of Arches you are starting in *Arches for Science* or *Arches HER*. In order to switch between versions of Arches in *Arches for Science* and *Arches HER*, use git to checkout a branch with the desired version of Arches. For example, to run *Arches for Science* using Arches 7.4, use the dev/7.4.x branch in the repo: `git checkout dev/7.4.x`

Example Docker Code Repositories for Production

The following lists Docker repositories that set up Docker containers running Arches configured for production deployment. These repositories are not officially part of the Arches Project, and may not have received the same level of review and vetting as Arches Project repositories. While these are not yet fully vetted, they can be a useful starting point or guide to use Docker for production deployment of Arches:

1. **arches-via-docker**

This repo (<https://github.com/opencontext/arches-via-docker>) uses Docker to provision containers running Arches (the most current stable version) and containers for the dependency PostgreSQL, ElasticSearch, and Redis services. It also starts an Nginx (as a proxy server) container as well as other containers to obtain and update SSL (for secure HTTPS) encryption certificates. You can use this repo directly or use it as a guide to see how to Arches can be configured for “production” deployments.

Understanding 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 or CSS 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.

To create a project, see *Creating a New Arches Project* in the installation guide.

Project Structure

The general structure of a new Arches project is::

```
my_project/  
├─ manage.py  
├─ my_project/  
│   ├─ settings.py  
│   ├─ datatypes/  
│   ├─ functions/  
│   ├─ media/  
│   ├─ templates/  
│   └─ widgets/
```

Not all files are shown

Important: At this level, “projects” are completely different from the mobile data collection “projects” that are mentioned elsewhere in this documentation.

settings.py

Many project-specific settings are defined here. You should use `settings_local.py` to 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, functions, and widgets

These directories will store the custom extensions that you can create for the project. Developers interested in pursuing these customizations should start with [Creating Extensions](#).

Understanding 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

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
```

-db	<i>true</i> to run <code>setup_db</code> to rebuild your database. default = 'false'
-ow	<i>overwrite</i> to overwrite concepts and collections. default = 'ignore'
-st	<i>stage</i> to stage concepts and collections. default = 'stage'
-s	a path to a zipfile located on github or locally
-o	operation name
-y	accept defaults (will overwrite existing branches and system settings with those in the package)
-bulk	uses <i>bulk_save</i> methods which run faster but don't call an object's regular <i>save</i> method
-dev	loads three test users

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.

If you pass the `-bulk` argument, know that any resource instances that rely on functions to dynamically create/edit tiles will not be called during package load. Additionally, some logging statements may not print to console during import of reference data. Whereas the default *save* methods create an edit in the edit history for each individual tile created, `-bulk` will instead create a single edit for all tiles, of type: "bulk_create". Resource creation will still be individually saved to edit history.

Note: It is important to note that you cannot load a package directly into core Arches. Packages must be loaded into 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.

1. Uninstall arches from your virtualenv

```
pip uninstall arches
```

2. Navigate into arches root folder delete the *build* directory
3. Reinstall arches

```
python setup.py install
python setup.py develop
```

4. Navigate to where you want to create your new project and run:

```
arches-project create mynewproject
```

Note: You can use the option `[{-d|--directory} <directory_name>]` to change the directory your new project will be created in.

5. Finally run the *load_package* command using the project's *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 and loading the resource models of your current project into your new package.

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

```
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)

business_data/resource_views

sql views of flattened resource models

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.

preliminary_sql

sql files containing database operations necessary for your project.

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.

Configuring a Package

Two different files are used to define custom settings for your package.

- **package_settings.py**

The django settings relevant to your project not managed in system settings. For example, you may want to include your time wheel configuration and your analysis SRID settings in this file so that users do not have to add these settings manually to their own settings file after loading your package. **This file is copied into your project when the package is loaded.**

- **package_config.json**

This file allows you to configure other parts of the data loading process. For example, the order in which the business data files are loaded. Contents of this file may look like

```
{
  "permitted_resource_relationships": [],
  "business_data_load_order": [
    "a_LHD_Investigative_Activities_HM.csv",
    "LHD_Actors.csv",
    "LHD_Archive_Sources.csv",
    "LHD_Bibliographic_Sources.csv",
    "LHD_Heritage_Asset_Areas_PC.csv",
    "LHD_Heritage_Asset_Artefacts_HM.csv",
    "LHD_Organizations.csv",
    "Lincoln_Heritage_Asset_Monument.csv"
  ]
}
```

Updating an Existing Package

If you make changes to the resource models in your project you may want to update your package with those changes. You can do that with the *update_package* command:

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

-d	full path to the package directory you would like to update
-o	operation name
-y	accept defaults (will overwrite existing resource models with those from your project)

Bear in mind that this command will not update a package directly on Github. It will however update a package in a local directory that you have cloned from an existing package on Github or created yourself with the `create_package` command.

1.1.2 For Arches Administrators and Users

This section provides information on how to configure and administer Arches, as well a brief discussion on how to create, edit, delete and search resources in Arches.

Initial Configuration

This section guides you through completing some additional configurations of Arches *after* you have successfully installed and launched (started) a running instance of Arches.

Set Resource Display Names

You may find that new resources are named **Undefined** in your search results. This is because the **Resource Descriptor Function** has not yet been configured for your Resource Model. Follow these steps to configure it separately for each Resource Model in your database.

1. Go to Arches Designer > Resource Models (/graph)
2. In the list of Resource Models, follow Manage > Manage Functions
3. Select the Define Resource Descriptors function to add it to the Resource Model
4. Use the tabs to configure all three different descriptor templates.

Configure a Descriptor Template

To configure a descriptor, you must first choose what card in the Resource Model holds the data you want to display. Choose this card in the dropdown, and variables corresponding to each node in that card will be added to the template, demarcated with `< >`. Now you can rearrange these variables, delete some of them, and/or add text to customize the descriptor.

Example: Consider a Resource with a Name node value of **Folsom School** and Name Type node value of **Primary**.

Template	Result
<code><Name>, <Name Type></code>	Folsom School, Primary
Building Name: <code><Name></code>	Building Name: Folsom School

Important: After you define your descriptors, you must **Re-Index** to update all of the existing resources in your database. This could take a while, if you have a lot of resources (that's why it's best to do this step right away!).

If there are multiple instances of a given card in a Resource, the first one added will be used to create these descriptors. To manually change this, edit the Resource in question and drag the desired tile to the top of the list.

Warning: Any user with read access permission to a resource will be seeing these resource descriptors wherever it shows up in search results or on the map. If a card is intended to be hidden from any group of users, it should not be used in this function.

Types of Descriptors

There are three different descriptors that appear through the Arches interface.

Display Name

Shown in search results list title, and top of reports.

Display Description

Shown in search results list description.

Map Popup

Shown in popup that appears when a resource is clicked in the map.

Arches System Settings

See also:

This section covers configuration settings that are managed through a browser. You will likely need to update other *settings that are defined elsewhere*.

Arches System Settings Interface

These are the settings found at <http://localhost:8000/settings>.

Default 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. Aggregating search results into a hexagonal grid can greatly improve performance of the map user-interface because fewer geometric features need to be delivered to a client's browser.

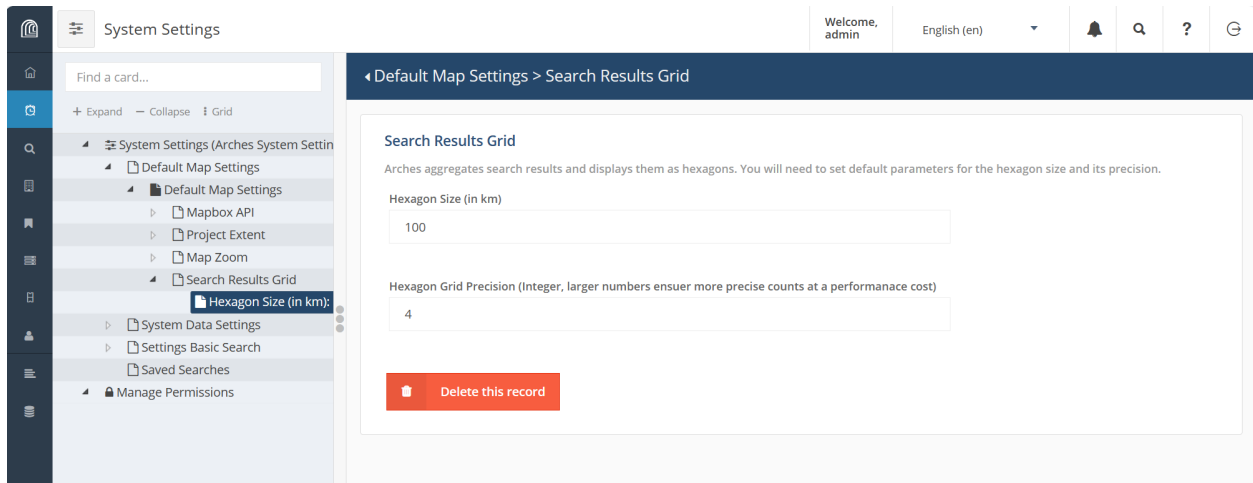


Fig. 1: Arches system settings for map search results grid

The Arches Elasticsearch component indexes a geohash of location data that powers efficient aggregation of geographic locations for resource instances. However, to enable map display of search results aggregated in a hexagonal grid, you first need to add a map layer that includes "source": "search-results-hex" in the layer definitions. You can read more about adding map layers (see [Creating New Map Layers](#)) or you can use SQL to insert a hex grid map layer as below:

```
INSERT INTO map_layers(maplayerid, name, ispublic, searchonly, sortorder,
layerdefinitions, isoverlay, icon, activated, addtomap)
VALUES (public.uuid_generate_v1mc(), 'Hex', true, true, 0, '[
{
  "layout": {},
  "source": "search-results-hex",
  "filter": [
    "==",
    "id",
    ""
  ],
  "paint": {
    "fill-extrusion-color": "#54278f",
    "fill-extrusion-height": {
      "property": "doc_count",
      "type": "exponential",
      "stops": [
        [
          0,
          0
        ],
        [
          500,
          5000
        ]
      ]
    }
  }
}]
```

(continues on next page)

(continued from previous page)

```

    },
    "fill-extrusion-opacity": 0.85
  },
  "type": "fill-extrusion",
  "id": "search-results-hex-outline-highlighted"
},
{
  "layout": {},
  "source": "search-results-hex",
  "filter": [
    "all",
    [
      ">",
      "doc_count",
      0
    ]
  ],
  "paint": {
    "fill-extrusion-color": {
      "property": "doc_count",
      "stops": [
        [
          1,
          "#f2f0f7"
        ],
        [
          5,
          "#cbc9e2"
        ],
        [
          10,
          "#9e9ac8"
        ],
        [
          20,
          "#756bb1"
        ],
        [
          50,
          "#54278f"
        ]
      ]
    },
    "fill-extrusion-height": {
      "property": "doc_count",
      "type": "exponential",
      "stops": [
        [
          0,
          0
        ]
      ]
    }
  ]
}

```

(continues on next page)

(continued from previous page)

```
        500,  
        5000  
    ]  
  ],  
  "fill-extrusion-opacity": 0.5  
},  
"type": "fill-extrusion",  
"id": "search-results-hex"  
}  
], true, 'ion-funnel', true, false);
```

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.

System Data Settings

Default Application Names

- **Application Name** - Name of your Arches app, to be displayed in the browser title bar and elsewhere.
- **Default Data Import/Export Name** - 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.

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.

Settings Basic Search

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). You can, however, modify the time wheel configuration using the advanced settings, [Time Wheel Configuration](#).

Maintaining Arches System Settings

Because these settings are stored in the database, as opposed to a `settings.py` file, if you drop and recreate your database, you will lose them and need to re-enter them by hand. To avoid this, you should run this command *after* you have finished configuring settings through the UI:

```
python manage.py packages -o save_system_settings [-d arches/db/system_settings]
```

A file named “System_Settings.json” will be saved to the directory indicated. If no directory is indicated the file will be saved to `settings.SYSTEM_SETTINGS_LOCAL_PATH`, which is `my_project/my_project/system_settings/` by default. This same path is used to import settings when a new package is loaded into your project.

Changing the Admin Password

The first item of business when preparing your production of Arches is to change the Admin user’s password. You cannot change the Admin user’s password in the Arches UI because the Admin account is not associated with an email. Instead you’ll need to use the Django admin page:

1. Login as admin to Arches or in the Django admin (<http://localhost:8000/admin/>)
2. Navigate to the Django admin user page <http://localhost:8000/admin/auth/user/>.
3. In the upper right of the page select **CHANGE PASSWORD** and follow the steps to update the password.

Settings - Beyond the UI

In reality, many more settings are used than are exposed in the UI. To see **all settings** look in the [core Arches settings.py file](#) (we try to leave comments on each one). The way these settings are cascaded through the app, and where they can be overwritten as needed, is described below.

Settings Inheritance

Settings can be defined in many different places. Here is the full inheritance pattern for a typical Arches project:

- **arches/settings.py**
If you installed Arches through pypi (`pip install arches`) this file will be deep in your virtual environment, and you shouldn't touch it.
↓ *values here can be superceded by...* ↓
- **my_project/my_project/settings.py**
Settings here define backend information specific to your app. For example, this is where you would add new references to template context processors.
↓ *values here can be superceded by...* ↓
- **my_project/my_project/package_settings.py (optional)**
Settings here define backend information specific to the package loaded to your app. You do not need to create or modify this file as it will be loaded when you load a package. However, you may want to edit this file if your intent is to design or modify a package.
↓ *values here can be superceded by...* ↓
- **my_project/my_project/settings_local.py (optional)**
Typically kept out of version control, a settings_local.py file is used for 1) sensitive information like db credentials or keys and 2) environment-specific settings, like paths needed for production configuration.
↓ *values here can be superceded by...* ↓
- **System Settings Manager**
Settings exposed to the UI are the end of the inheritance chain. In fact, these settings are stored as a resource in the database, and the contents of this resource is defined in the System Settings Graph. Nodes in this

graph with a name that matches a previously defined setting (i.e. in the files above) will override that value with whatever has been entered through the UI.

If you're a developer, you'll notice that the codebase uses:

```
from arches.app.models.system_settings import settings
```

in favor of:

```
from django.conf import settings
```

This is to ensure that UI settings are implemented properly. If you are using settings outside of a UI context you will need to follow the import statement with `settings.update_from_db()`.

Password Validators

By default, Arches requires that passwords meet the following criteria:

- Have at least one numeric and one alphabetic character
- Contain at least one special character
- Have a minimum length of 9 characters
- Have at least one upper and one lower case character

Admins can change these requirements by configuring the `AUTH_PASSWORD_VALIDATORS` setting in their projects `settings_local.py` file. Below is the default validator setting:

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'arches.app.utils.password_validation.NumericPasswordValidator',
        ↪ #Passwords cannot be entirely numeric
    },
    {
        'NAME': 'arches.app.utils.password_validation.SpecialCharacterValidator',
        ↪ #Passwords must contain special characters
        'OPTIONS': {
            'special_characters': ('!', '@', '#', ')', '(', '*', '&', '^', '%', '$'),
        }
    },
    {
        'NAME': 'arches.app.utils.password_validation.HasNumericCharacterValidator',
        ↪ #Passwords must contain 1 or more numbers
    },
    {
        'NAME': 'arches.app.utils.password_validation.HasUpperAndLowerCaseValidator',
        ↪ #Passwords must contain upper and lower characters
    },
    {
        'NAME': 'arches.app.utils.password_validation.MinLengthValidator', #Passwords_
        ↪ must meet minimum length requirement
        'OPTIONS': {
            'min_length': 9,
```

(continues on next page)

(continued from previous page)

```
    }  
  },  
]
```

To **remove a password validator** in Arches, you can simply remove a validator from the list of `AUTH_PASSWORD_VALIDATORS`.

To modify the list of **required special characters**, simply edit the list of characters in the `special_characters` option in the *SpecialCharacterValidator* validator.

To change the **minimum length of a password**, change the `min_length` property in the `MinLengthValidator` validator.

Advanced users can override or add new validators by creating their own validation classes as explained in [Django's password validation documentation](#).

Time Wheel Configuration

By default Arches will bin your data in the search page time wheel based on your data's temporal distribution. This enables Arches to bin your data efficiently. If your data spans over 1000 years, the bins will be by millennium, half-millennium and century. If your data spans less than a thousand years, your data will be binned by millennium, century, and decade.

You may decide, however, that the bins do not reflect your data very well, and in that case you can manually define your time wheel configuration by editing the `TIMEWHEEL_DATE_TIERS` setting.

Here is an example of a custom time wheel:

```
TIMEWHEEL_DATE_TIERS = {  
  "name": "Millennium",  
  "interval": 1000,  
  "root": True,  
  "child": {  
    "name": "Century",  
    "interval": 100,  
    "range": {"min": 1500, "max": 2000},  
    "child": {  
      "name": "Decade",  
      "interval": 10,  
      "range": {"min": 1750, "max": 2000}  
    }  
  }  
}
```

Each tier, ('Millennium', 'Century', 'Decade' are each tiers) will be reflected as ring in the time wheel. Properties:

- "name" - The name that will appear in the description of the selected period
- "interval" - The number of years in each bin. For example, if your data spans 3000 years, and your interval is 1000, you will get three bins in that tier.
- "root" - This applies only to the root of the config and should not be modified.
- "child" - Adding a child will add an additional tier to your time wheel. You can nest as deeply as you like, but the higher the resolution of your time wheel, the longer it will take to generate the wheel.
- "range" - A range is optional, but including one will restrict the bins to only those within the range.

If you do need to represent decades or years in your time wheel and this impacts performance, you can cache the time wheel for users that may load the search page frequently. To do so, you just need to activate caching for your project. If you have Memcached running at the following location `127.0.0.1:11211` then the time wheel will automatically be cached for the ‘anonymous’ user. If not you can update the CACHES setting of your project:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': os.path.join(APP_ROOT, 'tmp', 'djangocache'),
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    }
}
```

This will cache the time wheel to your project’s directory. There are other ways to define your cache that you may want to use. You can read more about those options in [Django’s cache documentation](#).

By default the time wheel will only be cached for ‘anonymous’ user for 24 hours. To add other users or to change the cache duration, you will need to modify this setting:

```
`CACHE_BY_USER = {'anonymous': 3600 * 24}`
```

The CACHE_BY_USER keys are user names and their corresponding value is the duration (in seconds) of the cache for that user. For example, if I wanted to cache the time wheel for the admin user for 5 minutes, I would change the CACHE_BY_USER setting to:

```
`CACHE_BY_USER = {'anonymous': 3600 * 24, 'admin': 300}`
```

Configuring Captcha

Setting up your captcha will help protect your production from spam and other unwanted bots. To set up your production with captcha, first [register your captcha](#) and then add the captcha keys to your project’s settings.py. Do this by adding the following:

```
RECAPTCHA_PUBLIC_KEY = 'x'
RECAPTCHA_PRIVATE_KEY = 'x'
```

Replace the x’s with your captcha keys.

Enabling User Sign-up

To enable users to sign up through the Arches UI, you will have to add the following lines of code to your project’s settings.py:

```
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = 'xxxx@xxx.com'
EMAIL_HOST_PASSWORD = 'xxxxxxx'
EMAIL_PORT = 587
```

Update the EMAIL_HOST_USER and EMAIL_HOST_PASSWORD with the correct email credentials and save the file. It is possible that this may not be enough to support your production of Arches. In that case, there's more information on setting up an email backend on the [Django site](#).

To configure what group new users are put into, add the following lines of code to your project's settings.py:

```
# group to assign users who self sign up via the web ui
USER_SIGNUP_GROUP = 'Crowdsourcing Editor'
```

If you would like to change which group new users are added to, replace 'Crowdsourcing Editor' with the group you would like to use.

Using Single Sign-On With an External OAuth Provider

To take advantage of single sign-on using an organization's identity provider, users can be routed through an external OAuth provider for authentication based on their email's domain.

Your arches application will need to use SSL and be configured with an application ID from your provider. This application ID will need to be configured with a redirect URL to your Arches application at auth/eoauth_cb, for example: https://qa.archesproject.org/auth/eoauth_cb

Once your application is set up with the provider, you can configure Arches to use it by updating EXTERNAL_OAUTH_CONFIGURATION, for example using an Azure AD tenant could look something like this:

```
EXTERNAL_OAUTH_CONFIGURATION = {
    # these groups will be assigned to OAuth authenticated users on their first login
    "default_user_groups": ["Resource Editor"],
    # users who enter an email address with one of these domains will be authenticated
    # through external OAuth
    "user_domains": ["archesproject.org"],
    # claim to be used to assign arches username from
    "uid_claim": "preferred_username",
    # application ID and secret assigned to your arches application
    "app_id": "my_app_id",
    "app_secret": "my_app_secret",
    # provider scopes must at least give Arches access to openid, email and profile
    "scopes": ["User.Read", "email", "profile", "openid", "offline_access"],
    # authorization, token and jwks URIs must be configured for your provider
    "authorization_endpoint": "https://login.microsoftonline.com/my_tenant_id/oauth2/v2.0/authorize",
    "token_endpoint": "https://login.microsoftonline.com/my_tenant_id/oauth2/v2.0/token",
    "jwks_uri": "https://login.microsoftonline.com/my_tenant_id/discovery/v2.0/keys"
    # enforces token validation on authentication, AVOID setting this to False
    "validate_id_token": True,
}
```

Accessibility Mode

As of version 7.5, Arches can be configured to meet [WCAG](#) defined **AA** level accessibility requirements for all public facing user interfaces (all content available to anonymous users without a login, including the home page, search interface, and resource reports). Improved accessibility helps to promote a more welcoming and inclusive community, and may help to meet important legal and ethical requirements, especially for institutions that serve the public.

To enable the “Accessibility Mode”, update your Arches project `settings.py` or `settings_local.py` file and add:

```
# Activate accessibility mode
ACCESSIBILITY_MODE = True
```

Once you’ve saved that change, restart Arches. Arches should now display more accessible user interfaces for public facing content. The specific accessibility enhancements activated in Accessibility Mode include:

- Markup to support labeling for screen readers
- Tabbing to support natural flow through the site
- Improved focus management especially when interacting with popup/slide out panels
- Updated drop downs to use an accessible version
- Updated text contrast
- Updated html to reflow properly on smaller screen sizes or when the screen is zoomed up to 400%
- Updated text sizes to use relative sizing

Administering

This section provides guidance on various aspects of database administration, data modeling, and permissions management.

Designing the Database

Arches Database Theory

Let’s begin with a brief primer on some of the core concepts upon which Arches is constructed.

Resources - Resources are what we call database records. If you are using Arches to create an inventory of historic buildings, each one of those buildings will be recorded as a “resource”. This terminology is used throughout the app.

Resource Models - When creating new Resources, a data entry user must decide which Resource Model to use, determining what information is collected for the Resource. Think of different Resource Models as categories of records in your database – “Buildings” vs. “Archaeological Sites” vs. “Cemeteries”, for example. Every Arches database must have at least one Resource Model.

Branches - Branches are tools for transport of complex node structures from one Resource Model to another. This allows you to avoid manually recreating the same “branches” in multiple Resource Models.

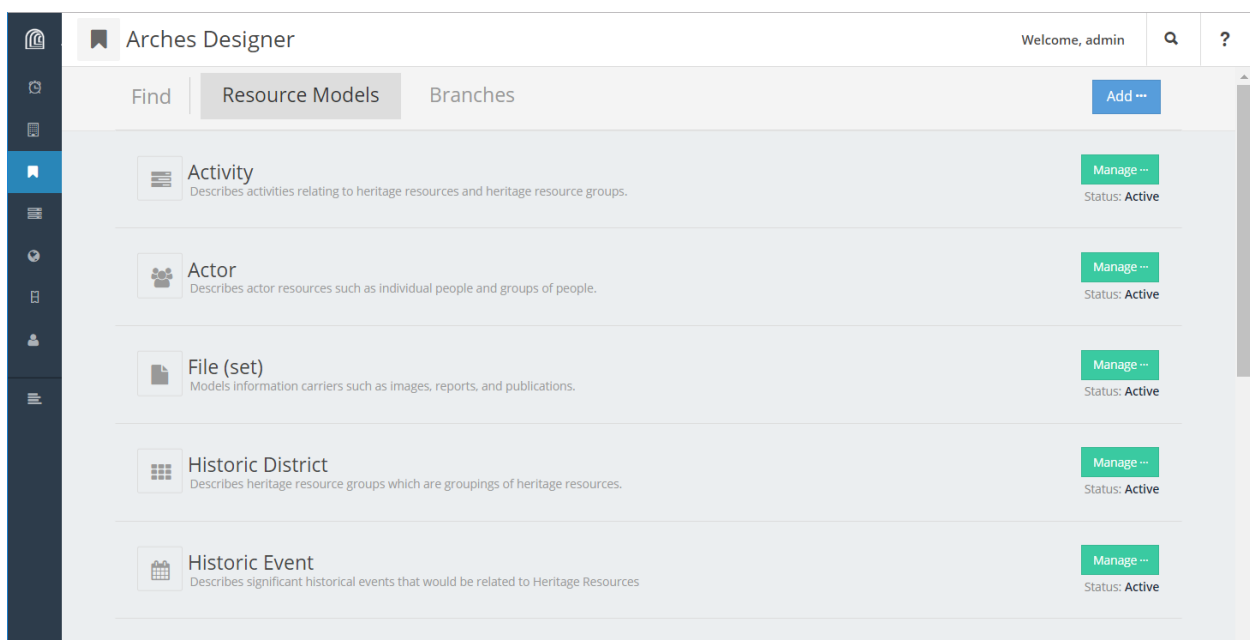
Note: Both **Resource Models** and **Branches** are sometimes referred to generically as “graphs”. This is because their underlying architecture is a graph. However, as you’ll see, they play completely different roles in Arches.

Important: The Arches Designer is used for altering the record-keeping structure of your database; it does not alter the physical *Data Model*.

Warning: If you need to have multiple versions of the same graph, perhaps multiple people are designing it or you need to retain earlier iterations while continuing to add nodes, you must Clone the graph. If a graph is renamed, exported, and imported, it will still overwrite the original, because the unique ID will remain unchanged.

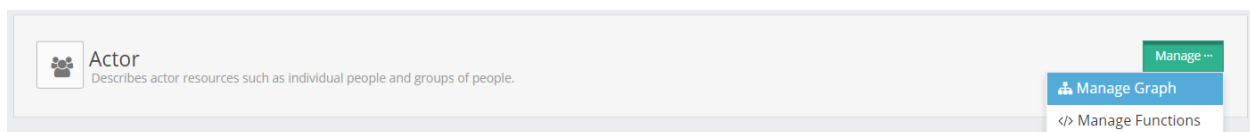
Arches Designer

The Arches Designer is where you export, import, duplicate, modify, and create your Resource Models and Branches. Any user who is part of the Graph Editor group will have access to the Arches Designer.



If you don't see any Resource Models listed in your Arches Designer, you may want to consider *loading a package*. Alternatively, you can directly import individual Resource Model files through **Add ...**.

To edit a Resource Model, click on it or click **Manage ...** > **Manage Graph** and you'll be brought to the Graph Designer.



Graph Designer

Almost all aspects of Resource Model and Branch design are handled in the Graph Designer. The exception is Functions, which are handled in the separate Function Manager.

The Graph Designer comprises three tabs, the *Graph Tab*, *Cards Tab*, and *Permissions Tab*. Each tab is used to configure a different aspect of the Resource Model: In the Graph Tab you design the node structure, in the Cards Tab you configure the user interface (card) for each nodegroup, and in the Permissions Tab you are able to assign detailed permission levels to each card. The general workflow for using the Graph Designer is to proceed through the tabs in that same order.

Graph Tab

The Graph Tab is where you build the actual graph, a structured set of nodes and nodegroups, which is the core of a Resource Model or Branch. As noted above, sometimes Resource Models and Branches are generically referred to as “graphs”, and this may seem confusing at first, but you’ll come to see that it is an appropriate nickname.

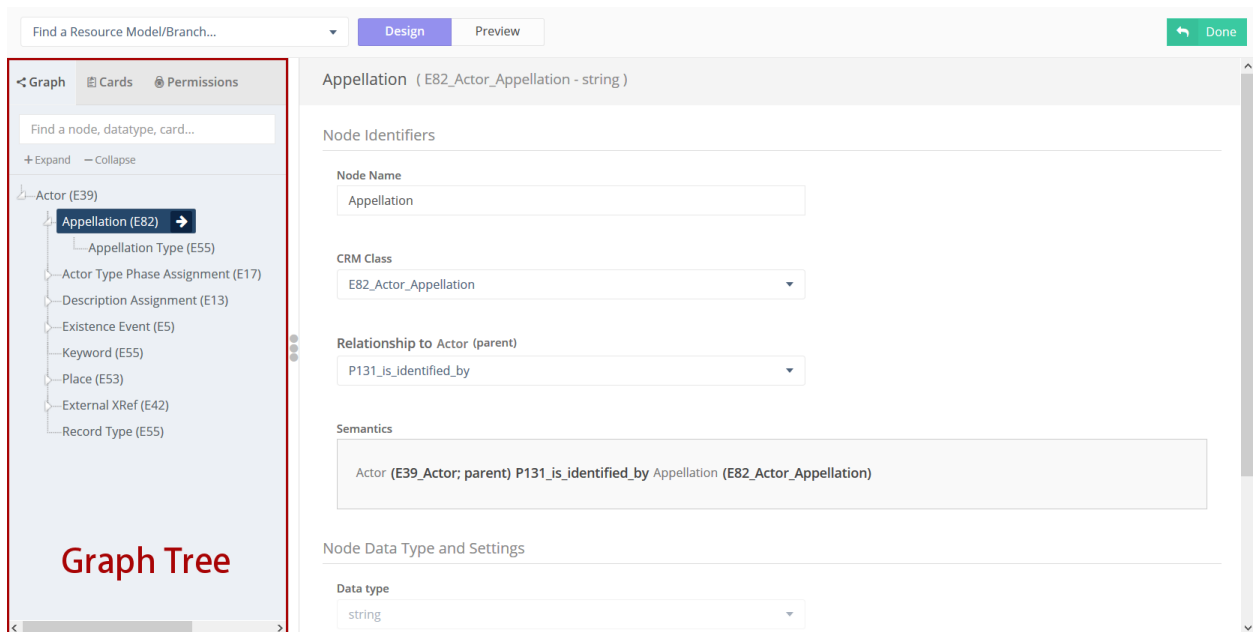


Fig. 2: Screenshot of the Graph Tab in the Graph Designer, showing an “Actor” Resource Model.

In practice, constructing the graph means adding nodes (or existing Branches) to the Graph Tree, which appears on the left side of the page when the Graph Tab is activated. When you add a new node, you set many different settings for that node, like datatype, in the main panel of the page.

During the graph construction process, you are able to create a new Branch from any portion of your graph. This is useful if you have completed a large section of the graph, and want to reuse it later in another Resource Model.

Note: If you are building a graph that uses an ontology, the ontology rules will automatically be enforced during this graph construction process.

Along the way, you can use the preview button to display the graph in a more graph-like manner. This view will be familiar to users of Arches going back to version 3.0.

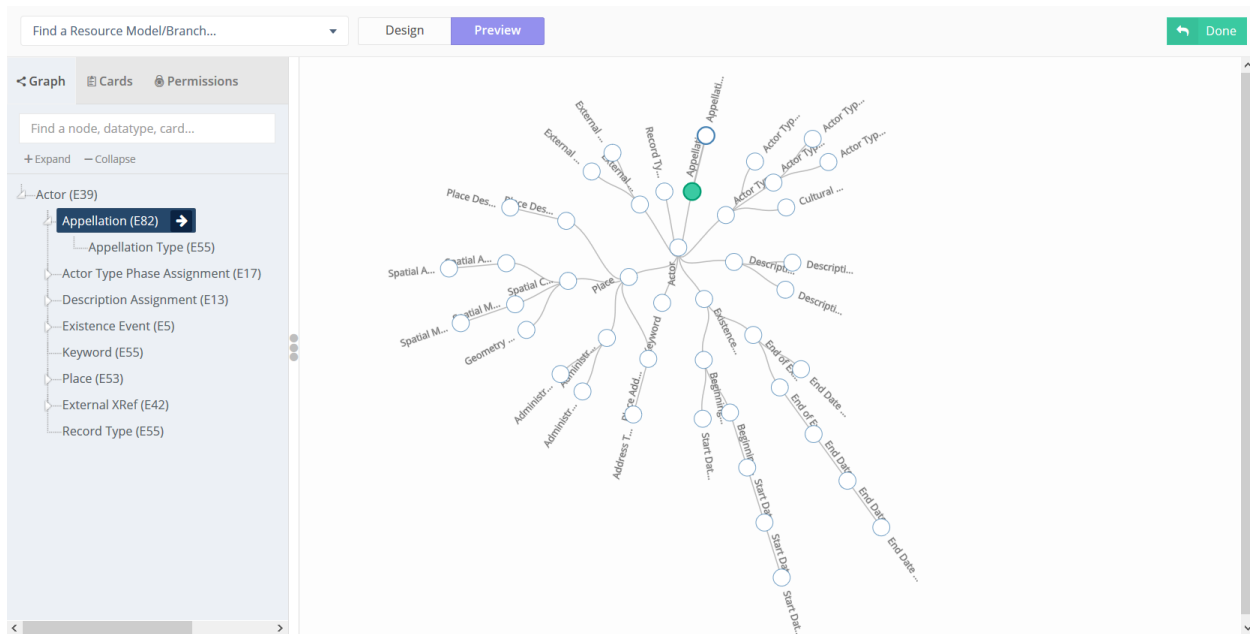


Fig. 3: Screenshot of the Graph Tab in the Graph Designer, showing the graph in preview mode.

Core Arches Datatypes

Nodes in Arches must be configured with a “Data Type”, and different datatypes store different kinds of information. For example, a **string** datatype is what you should use to store arbitrary text, like the name or description of a resource. A brief description of all datatype options in core Arches follows. Developers can extend Arches by *creating their own custom datatype*.

semantic

A semantic node **does not store data**. Semantic nodes are used where necessary to make symbolic connections between other nodes, generally in order to follow ontological rules. The top node of every graph is a semantic node.

string

Stores a string of text. This could be something simple like a name, or more something elaborate like a descriptive paragraph with formatting and hyperlinks.

number

Stores a number.

file-list

Stores one or more files. Use this to upload images, documents, etc.

concept

Stores one of a series of concepts from the Reference Data Manager. Users will choose a concept in a dropdown list or set of radio buttons. You'll further be prompted to choose a Concept Collection—this controls which concepts the user is able to choose from.

concept-list

Stores multiple concepts in a single node.

geojson-feature-collection

Stores location information. Use this for a node that should be displayed as an overlay on the main search map.

domain-value

Similar to “concept”, choose this to present the user with a dropdown list or set of radio buttons. Unlike “concept” this dropdown menu will not come from your system-wide controlled vocabulary, but from a list of values that you must define here.

domain-value-list

Stores multiple domain-values in a single node.

date

Stores a CE calendar date. See etdf for BCE and fuzzy date handling.

node-value

Stores a reference to a different node in this graph. This would allow you to store duplicate data in more than one branch.

boolean

Use this to store a “yes”/”no” or “true”/”false” value.

edtf

Stores an Extended Date/Time Format value. Use this data type for BCE dates or dates with uncertainty. This datatype requires extra configuration to inform the database search methods how to interpret EDTF values. Data entry users can enter edtf dates using formats listed in the EDTF draft specification.

annotation

Used to store an IIIF annotation.

url

Stores a web address.

resource-instance

Embeds a separate resource instance into this node. For example, you could add a node called “Assessed By” to a condition assessment branch, and use this data type. This would allow you to associate an individual stored in your database as an Actor resource with a specific condition assessment. Note that this construction is different from making a “resource-to-resource relationship”.

resource-instance-list

Stores a list of resource instances in a single node.

Cards Tab

Once you have added nodes to the graph, you can switch to the Cards Tab to begin refining the user interface. As you can see, the graph tree is replaced with a “card tree”, which is very similar to what users will see when they begin creating a resource using this Resource Model.

The top of the card tree is the root of the Resource Model, and you’ll select it to configure the public-facing resource report. Below this, you’ll see a list of cards in the Resource Model, some of which may be nested within others. There will be a card in the card tree for every nodegroup in the graph tree. Finally, within each card you’ll see one or more widgets. These correspond to nodes in the graph that collect business data. In the image above, the Appellation widget is selected.

When you select a card or a widget, you will see the Card Manager or Widget Manager appear on the right-hand side of the page. This is where you will update settings like labels, placeholder text, tooltips, etc. The middle of the page shows a preview of how a data entry user will experience the card.

Tip: While working with the Cards Tab, you may need to go back and change a node in the Graph Tab. Be aware that though you may expect node changes in the Graph Tab to cascade to widget configurations in the Cards Tab, this does

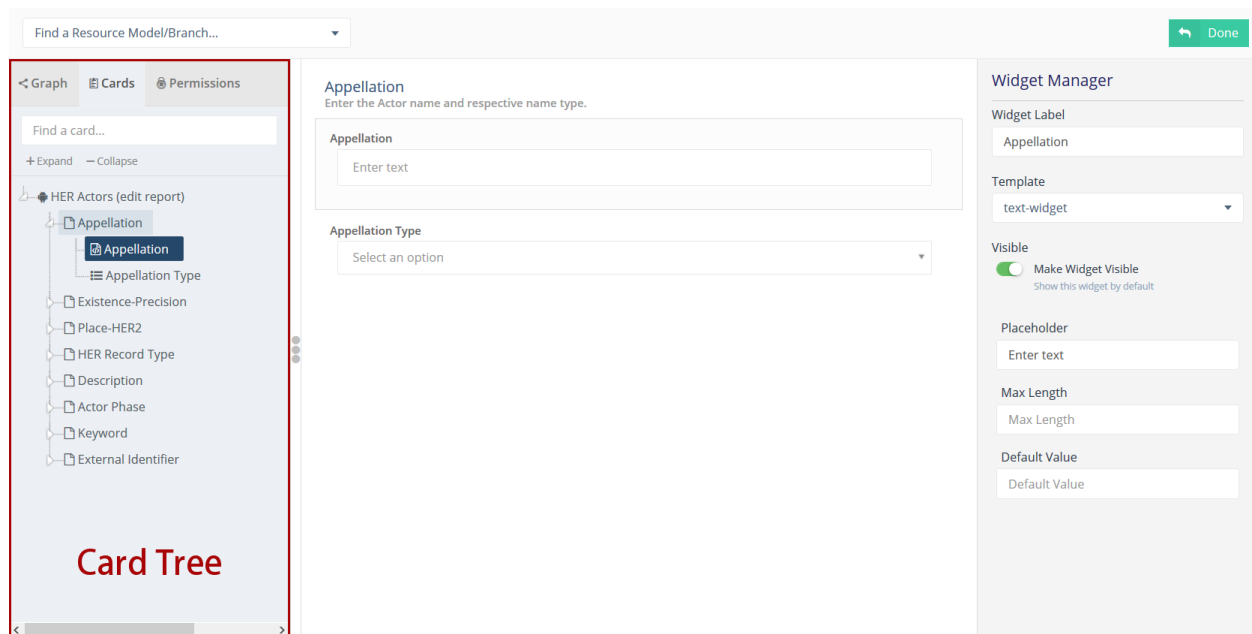


Fig. 4: Screenshot of the Cards Tab in the Graph Designer, showing an “Actor” Resource Model.

not always happen. Be sure to double-check your work!

Card Types

The UI of a card can be configured using a card component. Note that when you click a node in the card tree, the “Card Configuration” panel on the right-hand side of the screen will show the card component in a dropdown called “Card Type”.

The “CSS Classes” input box enables a user to enter space-separated class names (e.g. `card-empty-class card-incomplete-class`) that correspond to class names defined by a developer in `package.css`.

While card components can be created from scratch, Arches (v5 on) comes with a few out of the box:

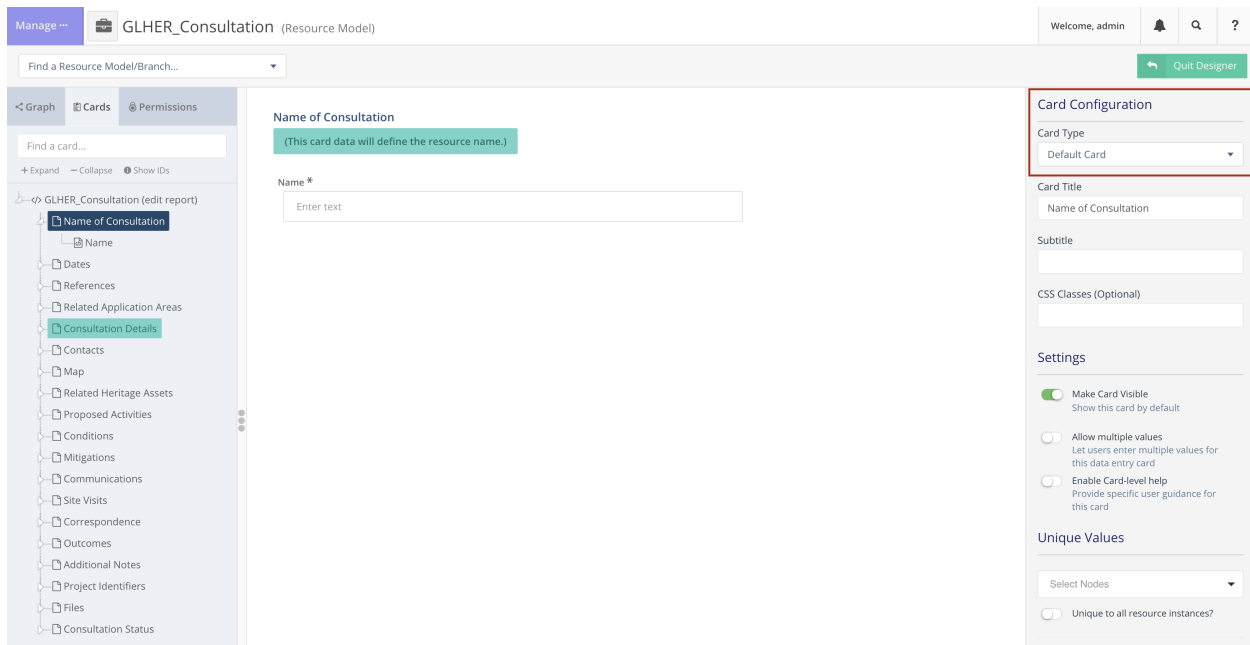


Fig. 5: Screenshot of the card manager user interface, highlighting “Card Type” dropdown in the top-right corner.

Grouping Card

The **Grouping Card** groups multiple cards into a single user interface (UI). One card acts as the root of the group by changing its “Card Type” to “Grouping Card” and then assigning “sibling” cards to it (in the last field of the Card Configuration section). While arches makes it easy to edit an existing card to include other nodes, the grouping card might be useful for cases where resource instances already exist for a model thus preventing you from editing the cards but you still want to group different cards together.

Map Card

The **Map Card** enables more customization for nodes of type `geojson-collection`. It has optional settings to start the map at a specific LatLng center and default zoom level. It can also import a particular map source layer of data into the UI. This might be useful if the user entering new geometry would benefit from having other resource data for reference in the map. To add a `map source` or `source_layer` simply type its name (no quotes).

Related Resources Map Card

The **Related Resources Map Card** enables a more rich user experience for nodes of type `related-resource`. Like the Map Card, map layer data representing resources can be added to a map UI such that the user can navigate geographically to select a related resource instead of paging through the dropdown list of relatable resources (however the dropdown still works normally in this card component). This card component is very useful if a user knows the geographic context of a resource (like what neighborhood it’s in) instead of its name. The steps to add such map data are the same as in the Map Card configuration panel.

Welcome, admin

Quit Designer

Card Configuration

Card Type
Map Card

Card Title
map rich

Subtitle

CSS Classes (Optional)

Select drawings map source (optional)

Select drawings map source layer (optional)

Select drawings text (optional)

Map Center Longitude
-0.088352147666961

Fig. 6: Screenshot of card configuration panel, highlighting the fields: “Select drawings map source” and “Select drawings map source layer”.

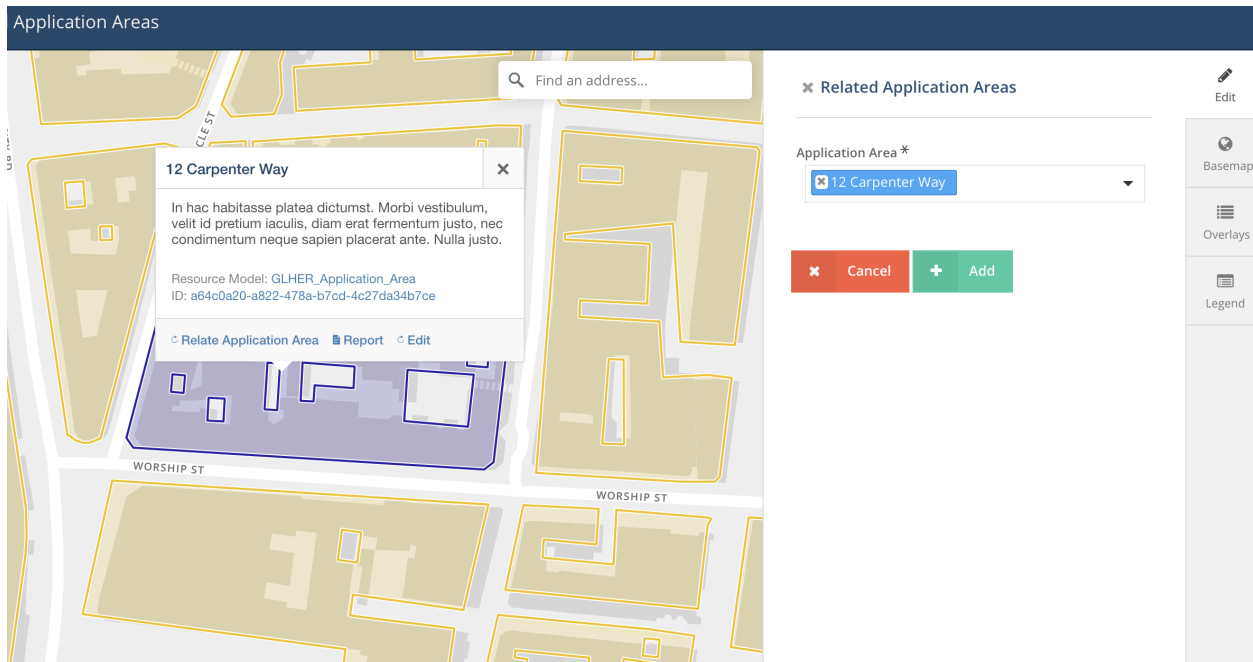


Fig. 7: Screenshot of a card using related resources map card, showing a selected resource in the map, polygon outlined in purple to show selection, and the resource instance's name selected in the dropdown widget to the right of the map.

Permissions Tab

Arches allows you to define permissions at the card level, so in the Permissions Tab you'll see the card tree, just as in the Cards tab. However, you will only be able to select entire cards, not individual nodes.

Once you have selected one or more cards, you can select a user or user group and then assign one of the following permissions levels:

Delete

Allows users to delete instances of this nodegroup. Note, this is not the same as being allowed to delete an entire resource, permissions for which are not handled here.

No Access

Disallows users from seeing or editing instances of this nodegroup. Use this permission level to hide sensitive data from non-authenticated users (the public).

Read

Allows users to see this nodegroup's card. If disallowed, the card/nodegroup will be hidden from the map and resource reports.

Create/Update

Allows users to create or edit instances of this nodegroup. This provides the ability to let users edit some information about a resource, while be restricted from editing other information.

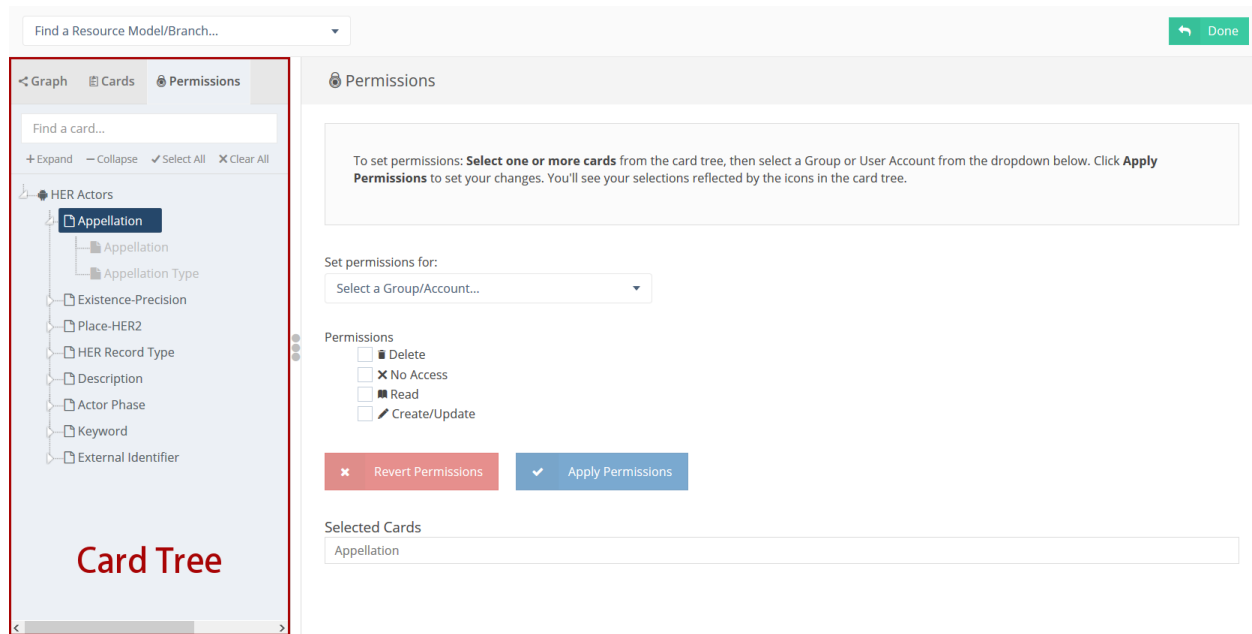


Fig. 8: Screenshot of the Permissions Tab in the Graph Designer, showing an “Actor” Resource Model.

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.

For many Arches applications data modelers will want to use a CRM (Conceptual Reference Model). The CIDOC CRM v6.2 is 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](#).

Loading an Ontology

Arches no longer comes preloaded with the CIDOC CRM, but it’s simple to load it or any other ontology. To load the CRM just download or clone it from this repository: <https://github.com/archesproject/cidoc-crm-ontology>. [download](#)

If you are developing an Arches package, you can simply unzip the downloaded zip file, and add the `cidoc_crm` folder to your packages ontologies directory. When you load your package, the CIDOC CRM will load with it:

```
/my_package/
├── ontologies
│   └── cidoc_crm
```

If you are not loading a package, you can unzip the downloaded file, and then run the following command with your virtual environment activated:

```
python manage.py load_ontology -s cidoc_crm
```

Loading a custom ontology

If you have created your own ontology or have a different version of the CIDOC CRM, then just add your files to a folder and include an *ontology_config.json* file which contains the metadata for your ontology. Here's an example:

```
{
  "base": "cidoc_crm_v6.2.xml",
  "base_name": "CIDOC CRM v6.2",
  "extensions": [
    "CRMsci_v1.2.3.rdfs.xml",
    "CRMarchaeo_v1.4.rdfs.xml",
    "CRMgeo_v1.2.rdfs.xml",
    "CRMdig_v3.2.1.rdfs.xml",
    "CRMinf_v0.7.rdfs.xml",
    "arches_crm_enhancements.xml"
  ],
  "base_version": "6.2",
  "base_id": "e6e8db47-2ccf-11e6-927e-b8f6b115d7dd"
}
```

You will need to generate a UUID to use as the `base_id`. Do not use the one in the example above.

Enforcing ontology rules

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.

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 can configure and add more.

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. New overlays can be added in the same manner as new basemaps.

Adding New Basemaps or Overlays

If you are a developer interested in creating new map layers (which could be new visualizations of resources or new basemaps and overlays), please see [Creating New Map Layers](#).

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

Permissions

As of Arches version 7.4.0, you can assign different permissions to specific Arches *users* and *groups*. To manage such permissions, please review [Map Layer Permissions](#).

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

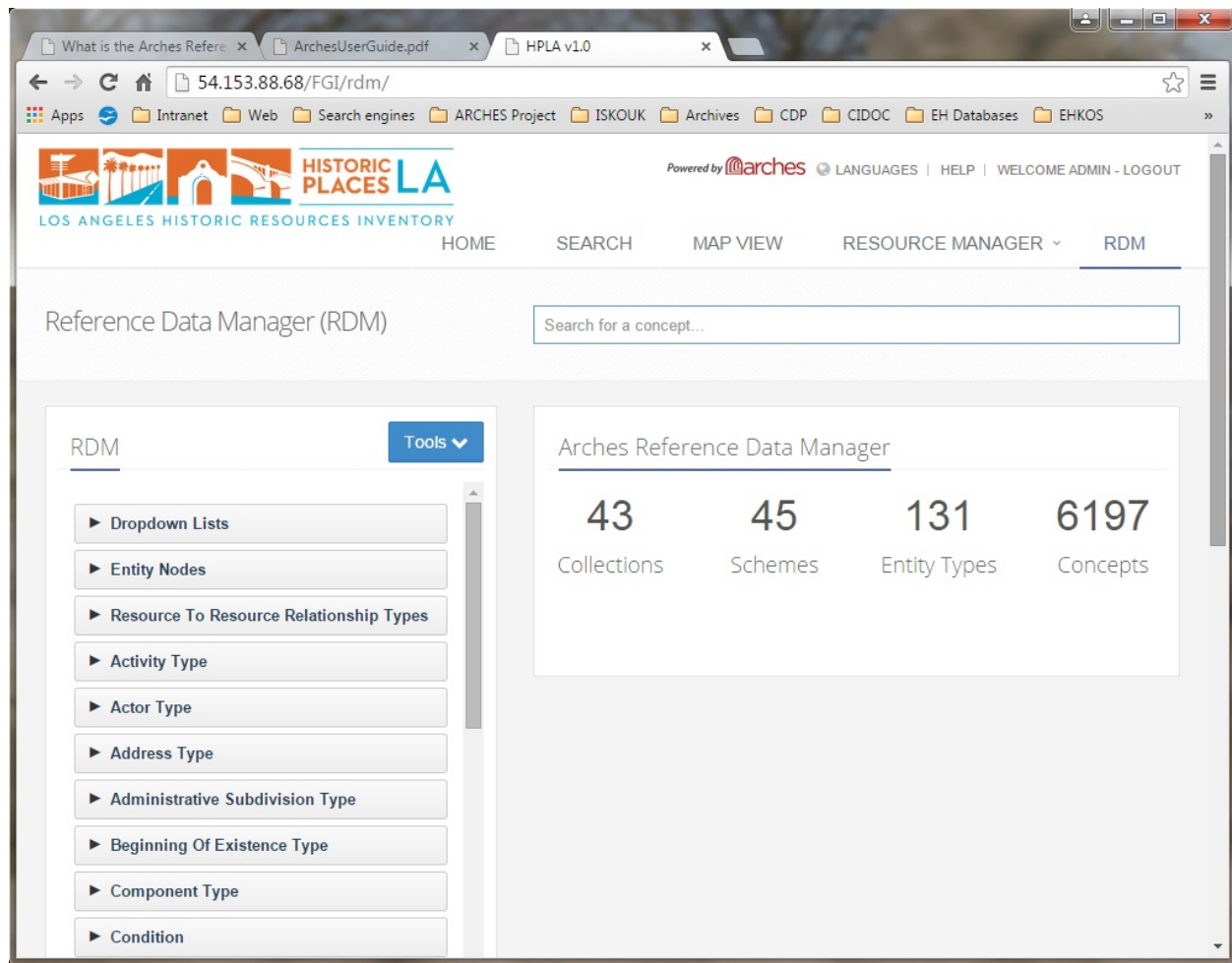
A concept scheme can be viewed as an aggregation of one or more concepts and the semantic relationships (links) between those concepts.

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.

Note: 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 be 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 care 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.

Note: 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.

Note: 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.

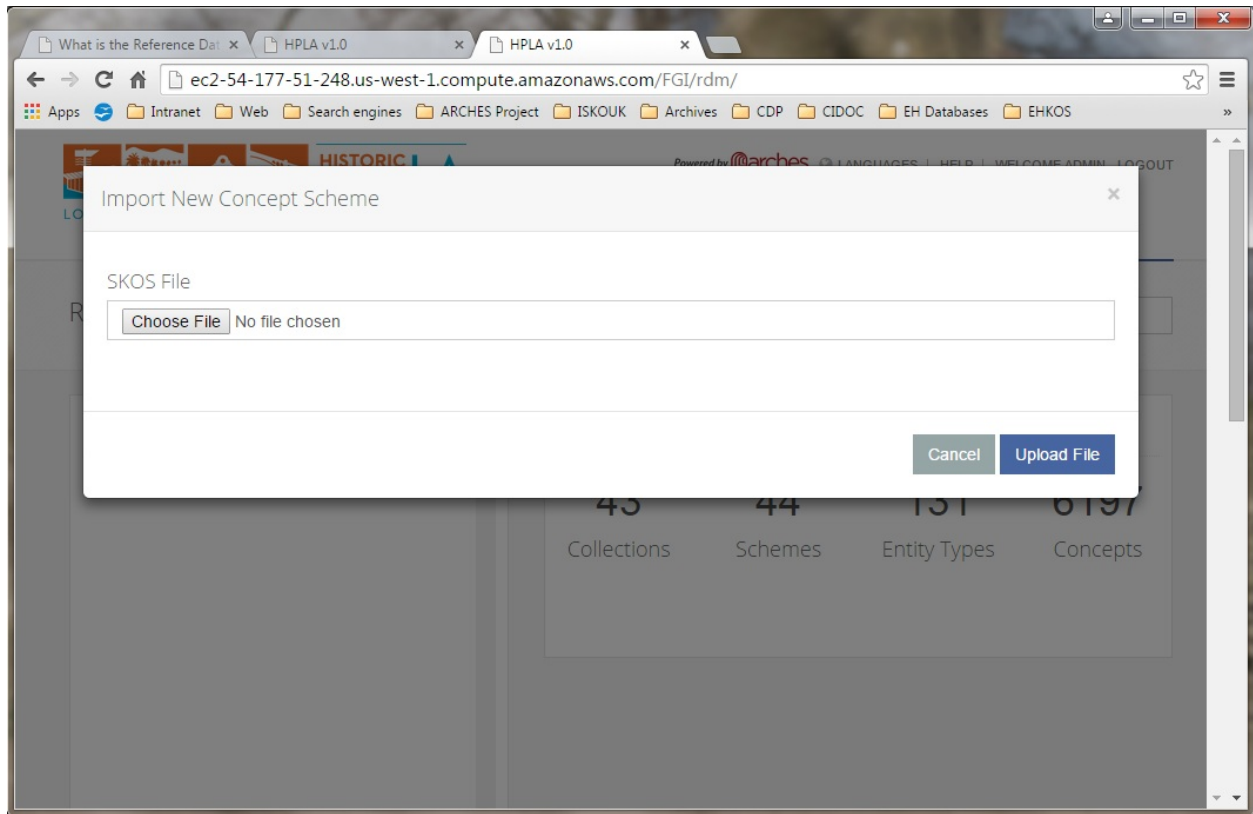
5. Click the **Yes** button. The concept is deleted (along with any of its children) and the node will disappear.

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.

Note: This file should be a SKOS file in any format parseable by Python's [RDFLib](#). Examples include RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, RDFa and Microdata.

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-compliant 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 structure.
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.

Note: 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.

Note: 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.

Graph Design, Instance Relationships, and Concept Labels

Arches v7.4.0 introduced features to enable administrators to define a wider range of relationship types between resources instances. Prior to v7.4.0, relationship types could only be defined in the resource instance widget using ontology properties. Arches v7.4.0 enables users to define relationships using concept values from concept collections managed in the *Reference Data Manager (RDM)*.

Steps to Make Custom Relationships between Resource Instances

You will need administrative privileges to use the RDM and edit resource models and branches. If you have such permissions, the following steps enable customization of relationships between resource instances:

- 1) **Create and define custom relationship concepts**

In the RDM Thesauri tab, navigate to and then select the “Arches” > “Resource To Resource Relationship Types” concept. Under the blue “Manage” option button (right side of the screen), select “Add Child”. Fill out the “Add Concept” form to describe your new custom resource to resource relationship type. If the direction of your custom relationship matters, you should also define an inverse relationship. For example, the inverse of “contains” can be “is contained by”.

- 2) **Add custom relationship concept to dropdown entry**

In the RDM Collections tab, navigate to and then select the “Resource To Resource Relationship Types” collection (it has the same name as the concept). Click the “Add dropdown entry” text, and this will open a dialogue where you can find and select your custom relationship concept to add to the “Resource To Resource Relationship Types” collection. This step makes your custom relationship available for use when you edit or create resource instances.

- 3) **Use your custom resource to resource relationship concept in a branch**

After you finish creating custom relationships (and their inverse relations), you can now use the Arches Designer to implement the custom relationships in your resource models. To use your custom resource to resource relationships, create a branch where the “Root Node Data Type” is either a “resource-instance” or a “resource-instance-list”. Once you select a resource model for use with this branch, click on the resource model label. This will open a form that will allow you to select a custom resource to resource relationship and the inverse of that relationship. After you save and publish, you will be able to use the custom resource to resource relationships as you create and edit resource instances.

Note: You can actually use any concept and collection you like (in other words, you are not restricted to the “Resource To Resource Relationship Types” concept). Our example use of the “Resource To Resource Relationship Types” concept and collection is a matter of convenience, and it probably makes sense for most users to use it unless their implementation really requires a more complex approach to defining relationships.

Django Admin User Interface

Arches is built with the Django framework [Django Documentation](#) and Arches makes use of the administrative user interface utilities that come standard with Django applications. The Django admin tools are intended for use by an organization's trusted internal management team. It's not intended for wider use by end users.

Arches administrators can use the the Django admin interface to control permissions for individual users and groups of users (see [Managing Permissions](#)) as well as certain site configurations and customizations (especially customizations for map interfaces).

Note: You can access the Django admin at `localhost:8000/admin`, the default admin credentials are `admin/admin`, which must be changed in production. In production, the URL to the Django admin interface will be `https://my-arches-site.org/admin/` **Any user with “staff” status can access the Django admin panel.**

Once logged into the admin panel, you'll see a page similar to this:

Bulk Data Manager

As of version 7.4, Arches provides **Bulk Data Manager** user interface tools for administrators to import and update large sets of data “in bulk”. These allow administrators to make changes across large sets of data, not just record by record.

Enable the Bulk Data Manager

The Bulk Data Manager is an Arches plugin (see [Plugins](#)). This plugin will be installed when you install Arches, but, by default, the Bulk Data Manager will be hidden.

To enable use of the Bulk Data Manager, login to the [Django Admin User Interface](#) and click the link to “Plugins” under models, click the “Bulk Data Manager”, and edit the JSON value for the attribute “Config”. To use the Bulk Data Manager, you will also need to enable Task Management (see [Task Management](#) and [Setting up Supervisor for Celery](#)).

To enable use of the Bulk Data Manager the Config should be: `{"show": true}`. To disable use of the Bulk Data Manager, the Config should be: `{"show": false}`. Once you've made your change, press the “Save” button in the lower right.

The image below illustrates how to enable the Bulk Data Manager:

Note: The Bulk Data Manager requires that you have properly installed and configured [Task Management](#) with Celery.

Using the Bulk Data Manager

Once you've enabled the Bulk Data Manager, Arches administrators will have access to Import, Edit, and Export functionality.

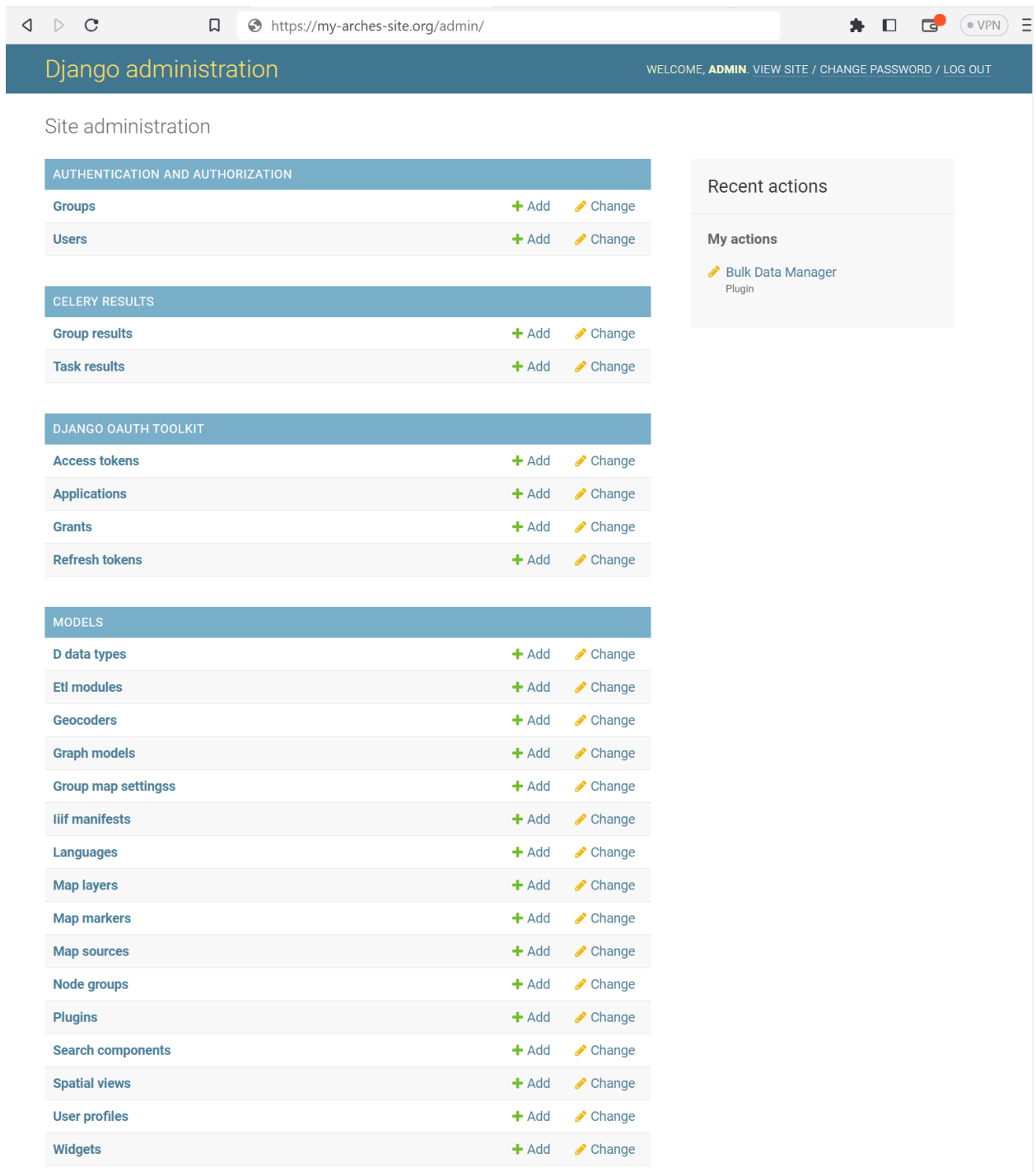


Fig. 9: Arches site administration in Django admin panel.

Fig. 10: Enable the Bulk Data Manager via the Django Admin panel.

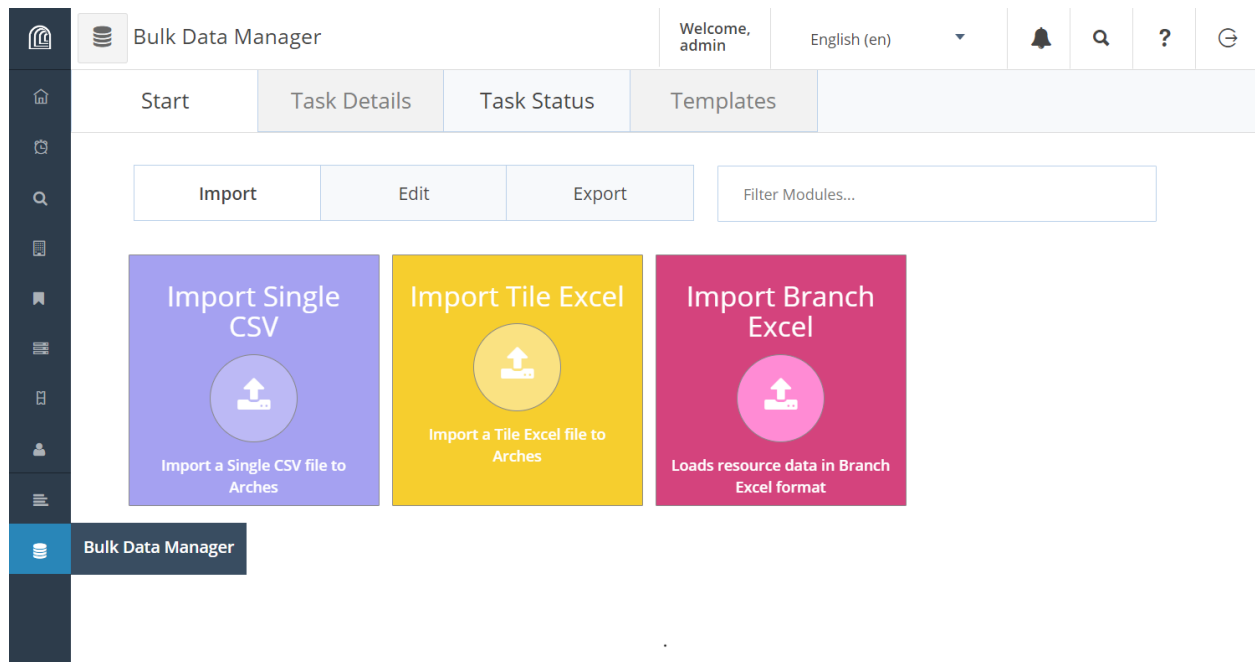


Fig. 11: Arches Bulk Data Manager plugin.

Import

The Bulk Data Manager has several **Import** related features to support the configuration and ingest of tabular organized data into Arches. These features presume familiarity with both the core Arches *Data Model* and the specific resource models and branches (see *Designing the Database*) used in your instance.

The Bulk Data Manager import tools support imports of data stored in CSV and Excel files. The CSV and Excel importers require that data in tables (and in the case of Excel, worksheets) will be organized according to map properly to your resource models and node structures for these resource models. To assist in creating data properly structured for successful import, you can download an Excel workbook template for a given resource model. The animation below illustrates how to export a template for an example resource model.

Fig. 12: Bulk Data Manager export of an Excel template for the (example) “Collection or Set” resource model

To describe how to use the Bulk Data Manager to import data, we’ll refer to the *Arches for Science* project *Collection or Set* resource model as an illustrative example. In the *Arches Designer*, the card for the *Name of Collection* branch of the *Collection or Set* resource model looks like this:

If you used the Bulk Data Manager to download an Excel template file for this *Collection or Set* resource model, you would see worksheets for each branch used with the resource model. The *Name of Collection* branch of the *Collection or Set* resource model has shaded nodegroups and nodes that looks like this:

The Excel template file also includes a worksheet called “metadata”. The metadata worksheet describes the datatypes (see more: *Core Arches Datatypes*) expected by each node:

Note: Bulk Uploading Files If you want to import resource instances that include datatype “file-list” nodes, then the files associated with those nodes will need to be imported along with the Excel workbook. To do this, zip compress a folder that includes the Excel workbook to be imported along with the associated files (such as image files) named in the “file-list” nodes. The files (such as image files) should be in the same folder as the Excel workbook (or the import CSV). The zip file should be named the same as the Excel workbook(or the import CSV). The Bulk Data Manager will

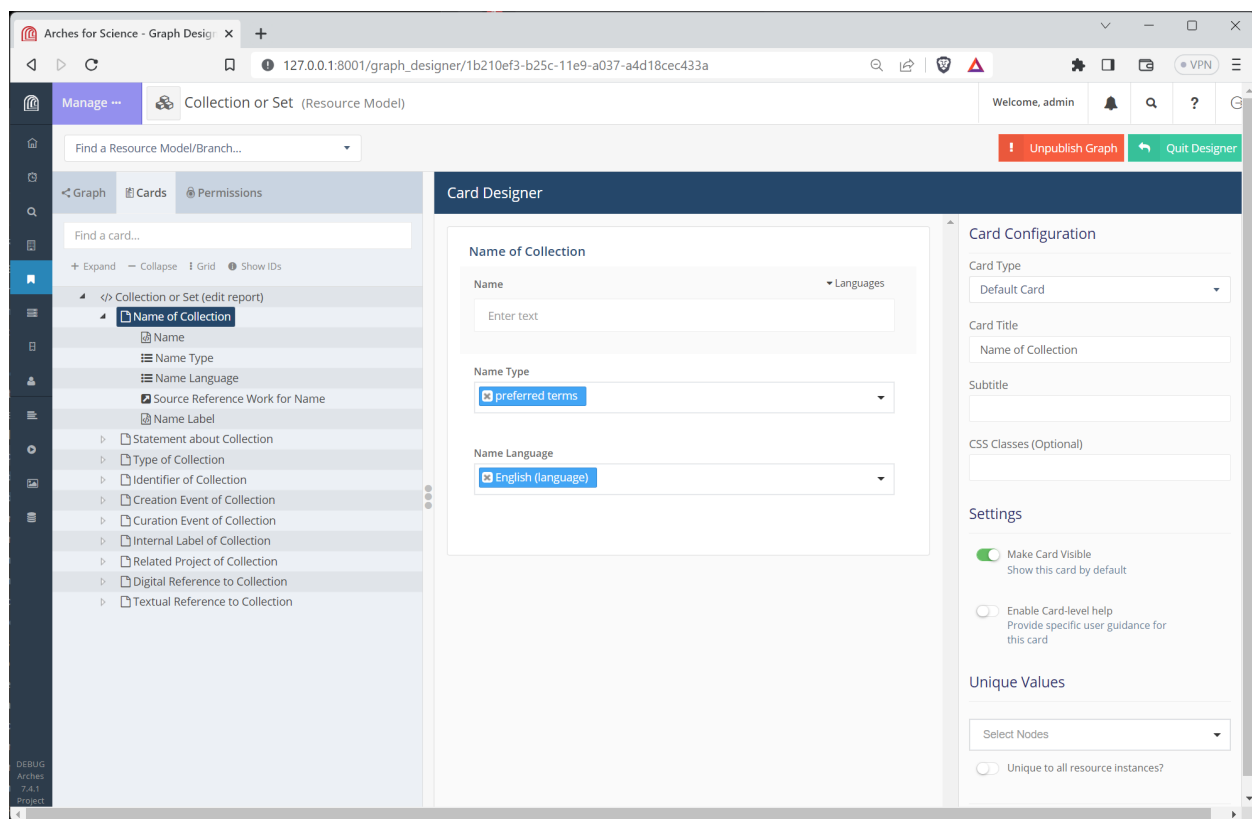


Fig. 13: Arches Designer view of the *Name of Collection* card used in the *Collection or Set* resource model

1.1. Table of Contents: Documentation Topics

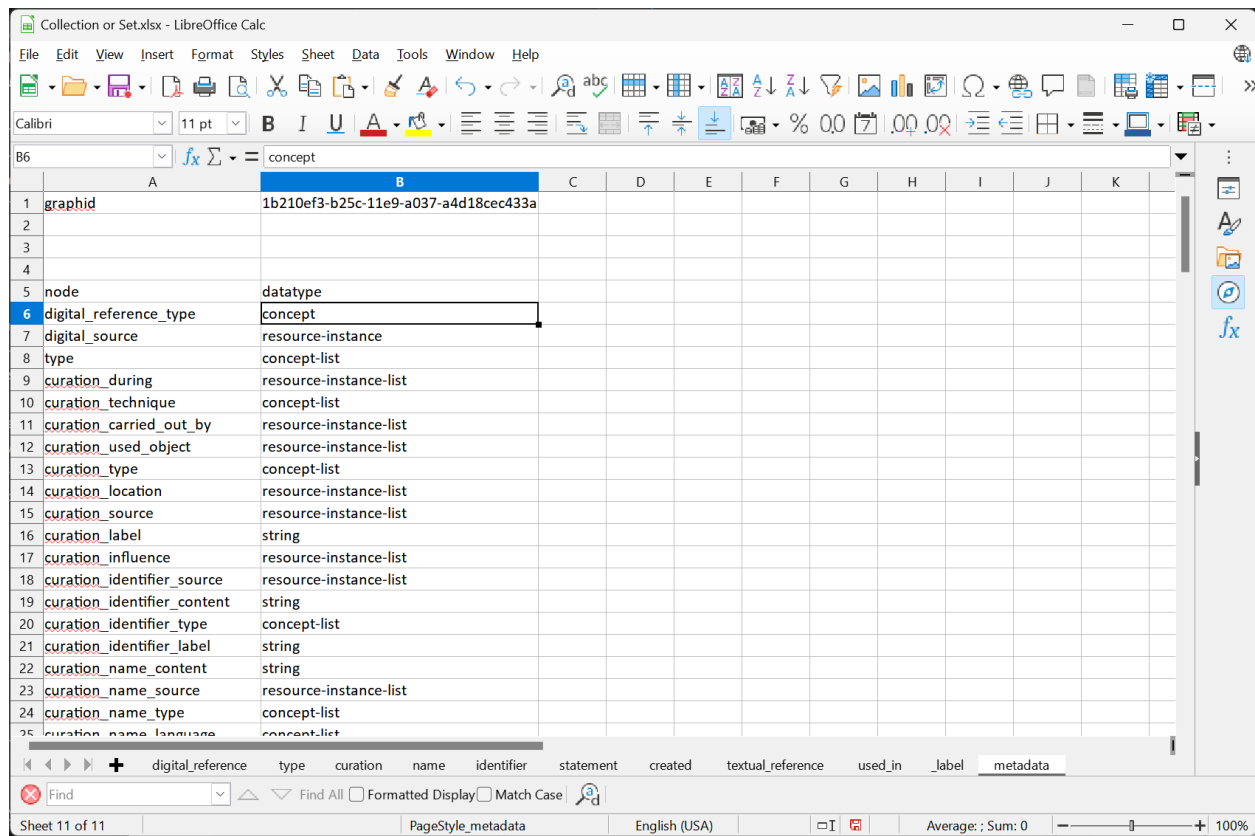


Fig. 15: Excel template *metadata* worksheet for datatypes used by *Collection* or *Set* branch nodes.

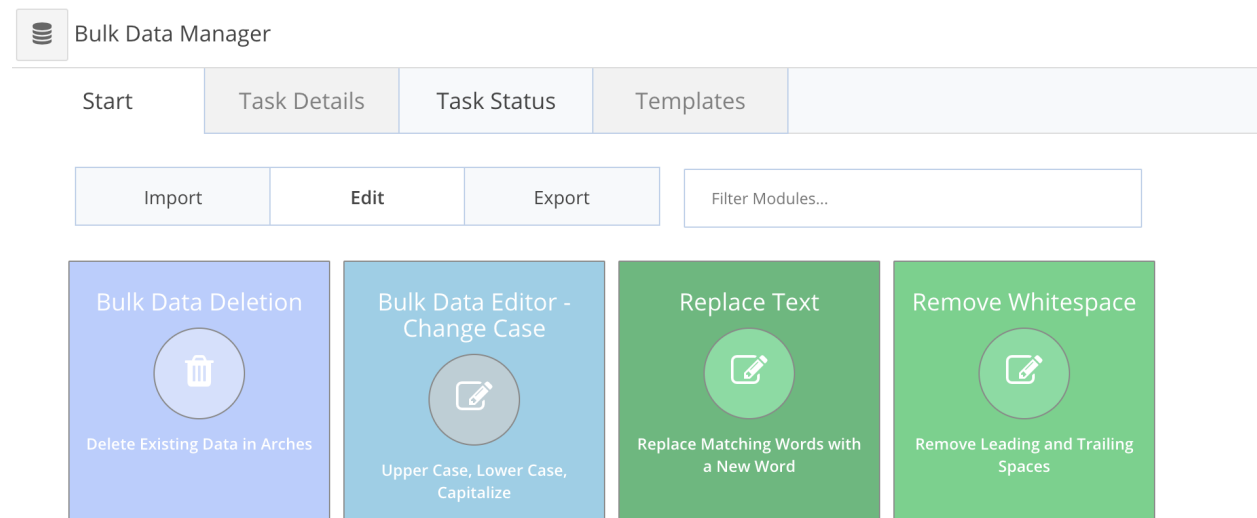
recognize the zip file and import the files along with the Excel workbook (or the import CSV). A valid zip file structure would look like this:

```
my_import.zip
├── my_import.csv
├── image1.jpg
├── image2.jpg
├── image3.jpg
└── image4.jpg
```

Edit

The **Edit** tab of the Bulk Data Manager enables Arches administrators to make mass edits of string data across many resource instances. As of version 7.5.0, the current string editing options include:

1. Bulk Deletion
2. Change case (uppercase, lowercase, capitalize)
3. Replace Text
4. Remove Whitespace



Editing operations require all or some of the following options:

1. Search URL (optional) - Defines the bounds of what resources can be edited. Actual edited resources could be less than what the search defines (see below).
2. Resource Model - Resource instances of the model to edit
3. Node - The node value in each resource instance to edit
4. Nodegroup - (Deletion only) the tile associated with the nodegroup to delete
5. Language - The language to update in each node

6. From and To - (Replace Text only) the text you would like to search and replace

Search URL details

Copy and paste a URL of a search that retrieves a set of resource instances that you want to limit your bulk edit operation to. This does not mean that those resources will actually be edited, only that resources that don't fall within that search result won't be edited.

For example, in a capitalize operation:

- If a search URL returns 3 records but one of them is already capitalized then only the remaining 2 uncapitalized records will be updated.
- If a search URL returns 3 records but the node in the model contains more than 3 records that are uncapitalized, then only the 3 records defined in your search will be updated.

Preview button

Once you're satisfied with the options you've selected click the preview button to preview a small set of records that match your criteria to see the before and after of the edit operation.

Start button

Click the start button if you'd like to actually kick off the edit operation. You will be taken to the Task Status tab. Depending on the operation selected and the number of resources being edited, this can take some time. Edit operations are placed into a work queue and at this point you can leave this page. The Task Status will update itself every 5 seconds (there is no need to refresh the page).

Preview

Showing first 5 of 637 values in 631 resources

Before	After
Sale recorded in catalog (record number 97758)	Sale Recorded In Catalog (Record Number 97758)
Sale recorded in catalog (record number 689070)	Sale Recorded In Catalog (Record Number 689070)
Sale recorded in catalog (record number 514543)	Sale Recorded In Catalog (Record Number 514543)
Sale recorded in catalog (record number 352812)	Sale Recorded In Catalog (Record Number 352812)
Sale recorded in catalog (record number 696989)	Sale Recorded In Catalog (Record Number 696989)

Start

Export

The **Export** tab of the Bulk Data Manager enables Arches administrators to make mass exports of resource instance data. The exported data will be in Excel workbooks. You choose to export data expressed in either a "Branch" or a "Tile" structure.

Fig. 16: Export of a resource instance data into an Excel workbook with the Branch structure

If you have resource instances that include datatype "file-list" nodes, then the files associated with those nodes will be exported into a zip file.

The Tile and Branch data export will export data in exactly the same formats used with the corresponding Bulk Data manager importers. This means that you can use the Bulk Data Manager to export data, make edits to the exported data, and then re-import the edited data. This can be useful for making mass edits to data that is not easily edited in the Arches user interface. The data made available through the Export tools will also provide invaluable examples of how to express data in a manner suitable for import.

resource_id	tileid	nodegroup	name	name_language	name_label	name_type	name_content	name_source
3ade3faa-c5c6-4a0f-aa29-0d91c9aa3d95	1fb734c5-745d-4e8f-8f4d-00c6f2c82fd0	name	bc35776b-996f-4fc1-bd25-9f6432c1f398f40c740-3c0a					
ad3623e8-9415-4d74-8615-94017064b861	00a51d02-2ede-421f-a0d7-ed9999064226	name	bc35776b-996f-4fc1-bd25-9f6432c1f398f40c740-3c0a					
475f79fb-4fe1-46f1-b3f8-60b03a513e79	f953a9b-8b43-433f-ae48-c2881db3cd80	name	bc35776b-996f-4fc1-bd25-9f6432c1f398f40c740-3c0a					

Fig. 17: Example “Branch” Excel export of resource instance data

Deleting Stuck Tasks

The **Bulk Data Manager** uses worker processes (see [Task Management](#)) to perform operations on the database. Occasionally, an operation may fail and result in a stuck task. If you have a task that is stuck and you want to delete it, you can do so via SQL operations on the database. The relevant tables relating to tasks are `load_event` and `load_staging`. Here is an example of how delete a record in the `load_event` table:

```
-- Verify the load_event of interest exists
SELECT * FROM load_event WHERE loadid = 'some-load-event-uuid';
```

Once you’ve verified the record exists, and this is a record you do want to delete, you can delete it with the following SQL:

```
-- Now delete the load_event record
DELETE FROM load_event WHERE loadid = 'some-load-event-uuid';
```

This may not immediately cause tasks to be removed from the **Bulk Data Manager** queue in the web UI, but it should clear stuck tasks.

Managing Permissions

Permissions in Arches are handled on a few different levels.

- *Managing Users and Groups in Django Admin*

Determine who can access what parts of Arches using the built-in Django admin interface.

- *Resource Model Permissions*

Determine which Users and Groups can read/edit/delete specific portions of a Resources Model.

- *Resource Instance Permissions*

Grant access to specific Resource instances on a per-User and/or per-Groups basis.

- *Media Permissions*

Restrict access to site media.

- *Map Layer Permissions*

Restrict access to map layers.

Managing Users and Groups in Django Admin

Arches is a complex platform, and some users must be able to access specific areas of the application while being restricted from others. This level of access is handled by adding Users to certain Groups through the Django admin interface.

Note: You can access the Django admin at `localhost:8000/admin`, the default admin credentials are `admin/admin`, which must be changed in production. **Any user with “staff” status can access the Django admin panel.**

Once logged into the admin panel, you’ll see this at the top of the page:

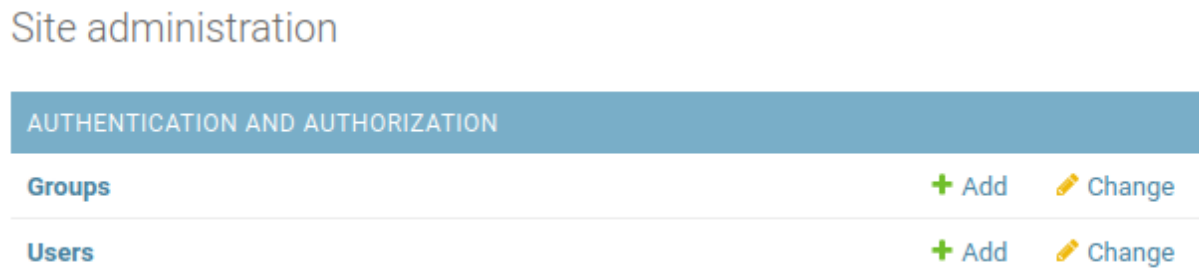


Fig. 18: Arches site administration in Django admin panel.

Click **Users** to see a list of all your Arches users. Selecting a user will yield a generic profile page like this:

In the “Permissions” section here there are three fields.

Active

This account is active and the user can log in. Unchecking this box allows you to retain a user account while disallowing them from accessing Arches.

Staff status

This user can access (and make changes within) the Django admin panel.

Superuser status

This user has full access to the entire Arches platform, and is considered a member of every Group.

Next, you’ll see where you can assign the user to different Groups. Arches comes with many default different groups, and each one gives its members access to different parts of the application. A user can be a member of as many different groups as needed.

Graph Editor

Use Case

For creating and testing branches and models.

Change user

Username:	<input type="text" value="cjmorton"/>
Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.	
Password:	algorithm: pbkdf2_sha256 iterations: 150000 salt: ulLDEc***** hash: ZG0VrM*****
Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form .	

Personal info

First name:	<input type="text" value="Chet"/>
Last name:	<input type="text" value="Morton"/>
Email address:	<input type="text"/>

Permissions

☒ **Active**
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☐ **Staff status**
Designates whether the user can log into this admin site.

☐ **Superuser status**
Designates that this user has all permissions without explicitly assigning them.

Fig. 19: User's admin profile

Access Privileges

Create/design graphs, branches, functions, and RDM. Add/edit business data with Resource Editor privileges. Unable to access system settings or mobile projects.

Resource Editor

Use Case

Ability to add/edit/delete provisional data more liberally than a Crowdsourcing Editor user.

Access Privileges

Add/edit/delete resources.

RDM Administrator

Use Case

Add/edit/manage RDM concepts

Access Privileges

Full access to the RDM - no access to the rest of Arches.

Application Administrator

Use Case

Control over Django admin page... can add/edit/delete users and user groups within Django admin console

Access Privileges

Has Django superuser status (see above) which gives it full access to Arches.

System Administrator

Use Case

Changing data stored in the system settings graph.

Access Privileges

Ability to access/edit data in Arches System Settings.

Crowdsourcing Editor

Use Case

Creation of provisional data from an untrusted source. Default group user is assigned to when first added to the system via e-mail sign-up.

Access Privileges

Add/edit/delete resources your own provisional data tiles

Guest

Use Case

Read-only access for anonymous users (non-authenticated users are automatically in this group)

Access Privileges

Read-only access to all business data

Resource Reviewer

Use Case

Review provisional data and promote it to authoritative

Access Privileges

Add/Edit authoritative business data. Ability to promote provisional data to authoritative.

Resource Exporter

Use Case

Control permissions to make exports of search result resource instances

Access Privileges

This group was added in Arches version 7.4.0. Membership in this group is now *required* to export resource instance data from search results. By default, the `anonymous` user is a member of this group. If you want to disable export of resource instance data from searches for anonymous users, remove the `anonymous` user from this group. Similarly, you can control resource instance export privileges for other users by adding or removing them from the `Resource Exporter` group.

Feel free to make new groups as needed, but do not remove any of those listed above. Groups are also used in other aspects of permissions as described below.

Resource Model Permissions

Permissions are applied to each card and by default, the guest user (aka anonymous user) has read privileges to all data. If you have data you do not want to share with all users, follow these directions when designing your database: [Permissions Tab](#).

Resource Instance Permissions

<https://github.com/archesproject/arches-docs/issues/218>

Media Permissions

If you want to ensure that all media file (uploaded photographs, etc.) access requires authentication, you can set `RESTRICT_MEDIA_ACCESS` to `True`.

Be aware that in doing so, all media file requests will be served by Django rather than Apache. This will adversely impact performance when serving large files or during periods of high traffic.

In `settings_local.py` add this line:

```
RESTRICT_MEDIA_ACCESS = True
```

Map Layer Permissions

As of Arches version 7.4.0, you can assign different permissions to specific Arches *users* and *groups*. To manage **Map Layer Permissions**, login to the [Django Admin User Interface](#) and click the link to “Map layers” under models, and then click on the specific Map Layer that you’d like to update for permissions.

To update permissions of a specific Map Layer, navigate to the *OBJECT PERMISSIONS* link in the upper right as illustrated below:

Note: You will **ALSO** need to make sure the `Ispublic` flag for the Map Layer is deactivated. That flag is located lower down, well below the the link to the *OBJECT PERMISSIONS*, see below:

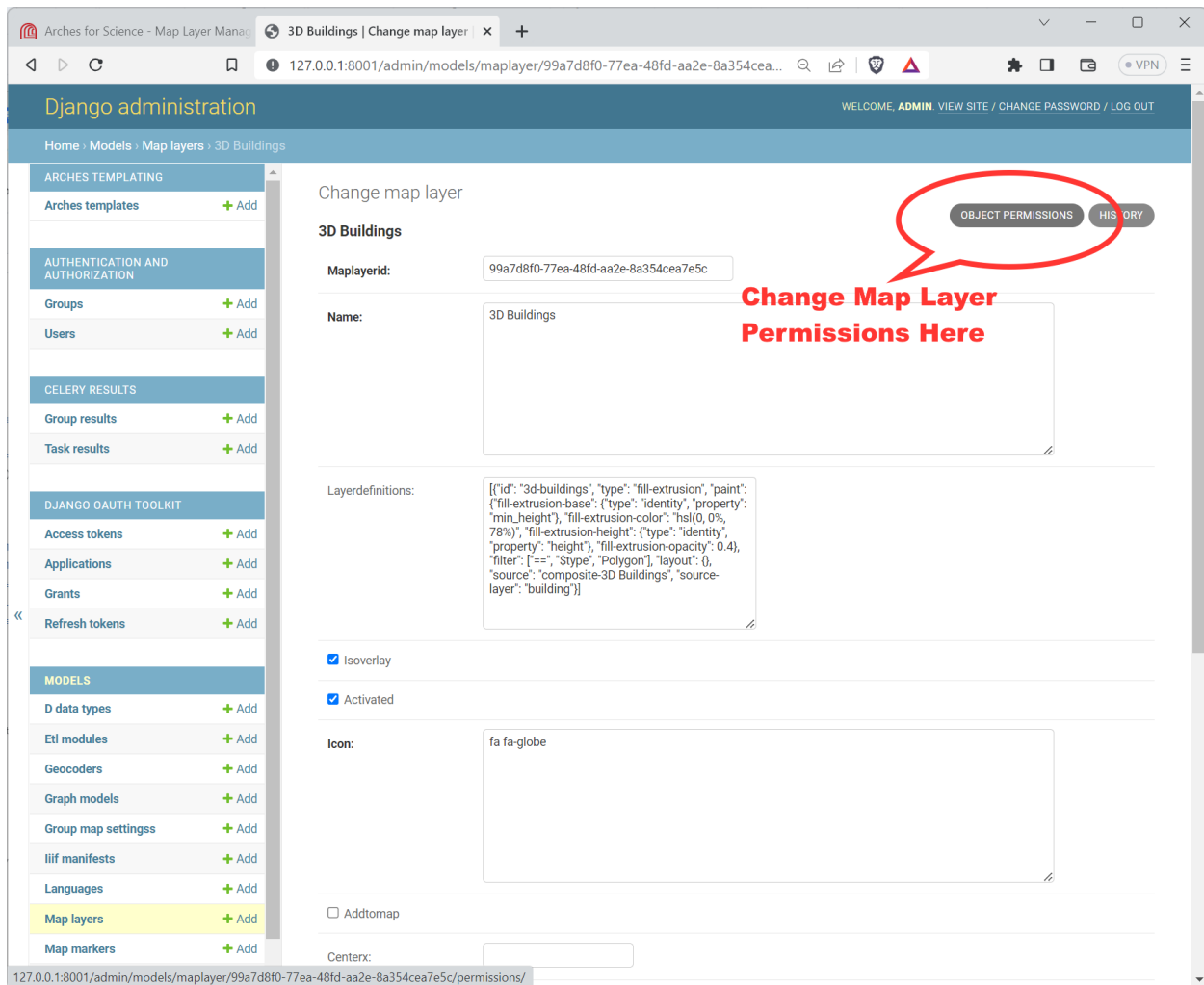


Fig. 20: Link to the Object Permissions update form for a Map Layer in the Django Admin panel.

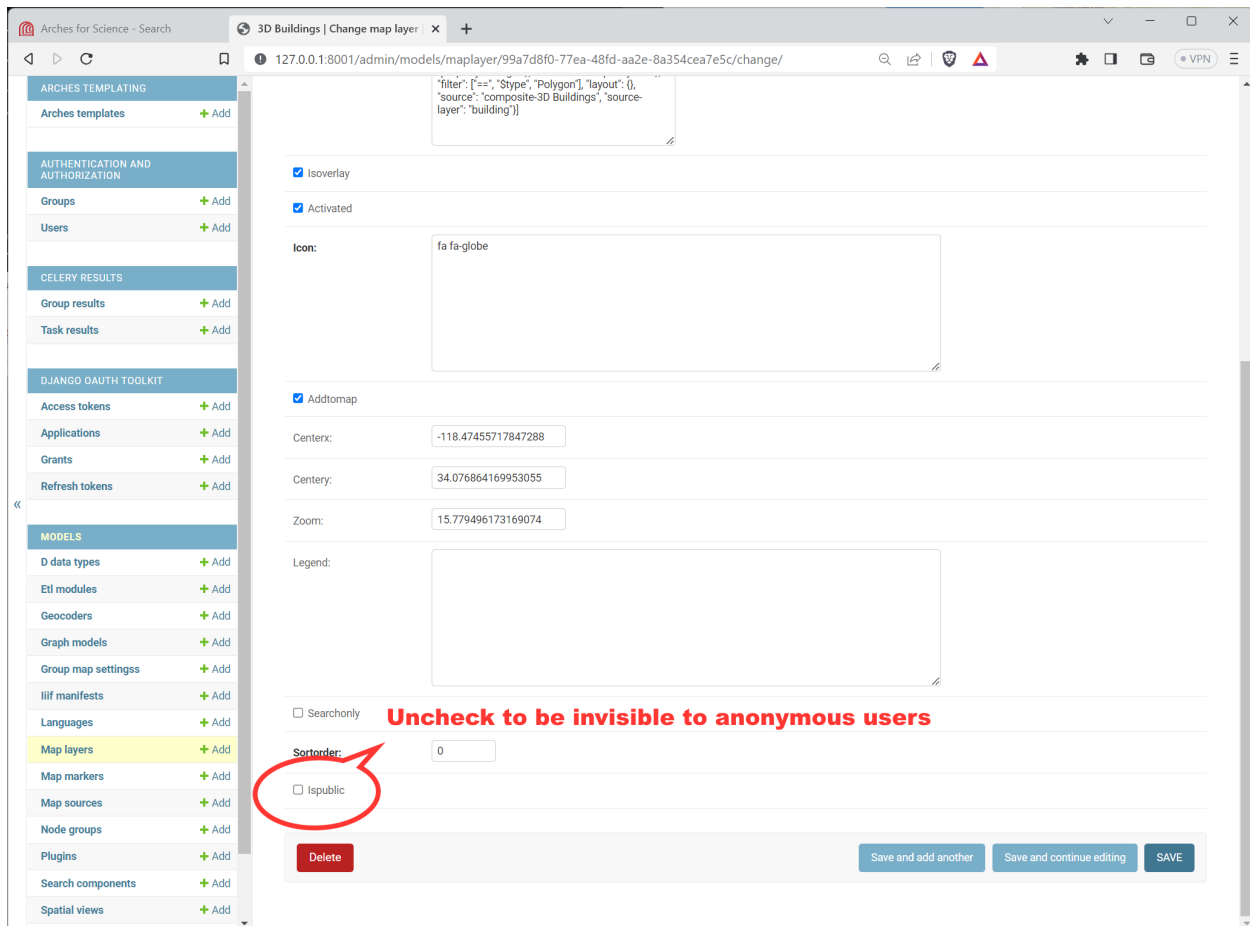


Fig. 21: Location of the `Is public` flag for a Map Layer in the Django Admin panel.

Once you click the *OBJECT PERMISSIONS* link, you will see a form that will let you name users (by their username) and groups, by their group name. Once you add the name for the user or group, press the “Manage user” or “Manage group” button as appropriate. See the illustration below for an example:

After clicking the “Manage user” or “Manage group” button, you will reach another form where you can add or subtract specific permissions for this user (or group) and Map Layer. See the illustration below for an example:

Once you have updated the permissions, it’s a good idea to test the Arches interface to make sure the permissions for the Map Layer are properly applied.

Spatial Views (preview)

Warning: This feature is in preview and therefore is not feature complete or may have some bugs.

As of Arches version 6.1.0, it is possible to create spatial views of resource instance data that can be consumed by any client that supports PostGIS spatial views.

Currently the preview only allows the spatial views to be created in Django admin by managing the Spatial Views entities.

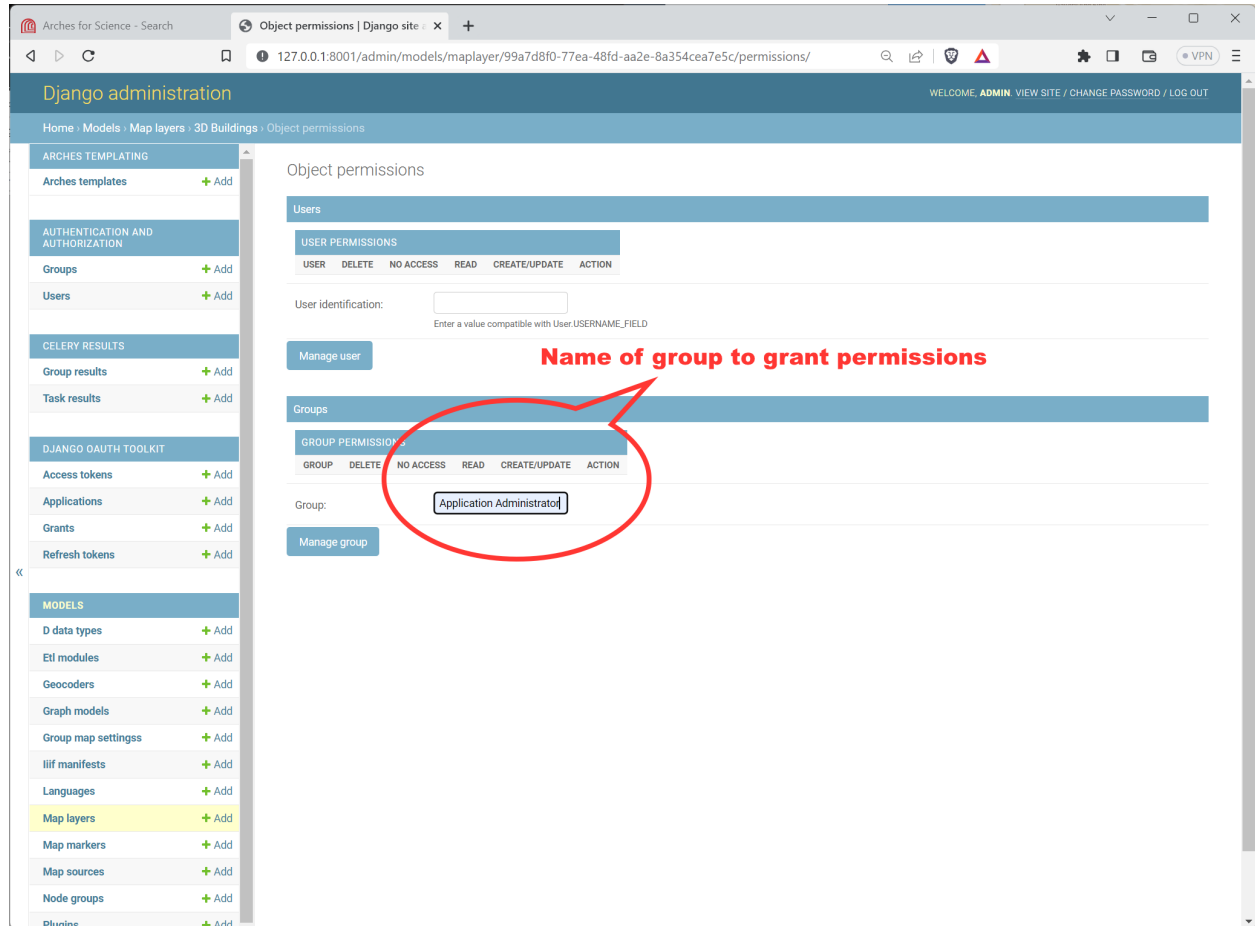


Fig. 22: Adding a Group Name to the Object Permissions for a Map Layer in the Django Admin panel.

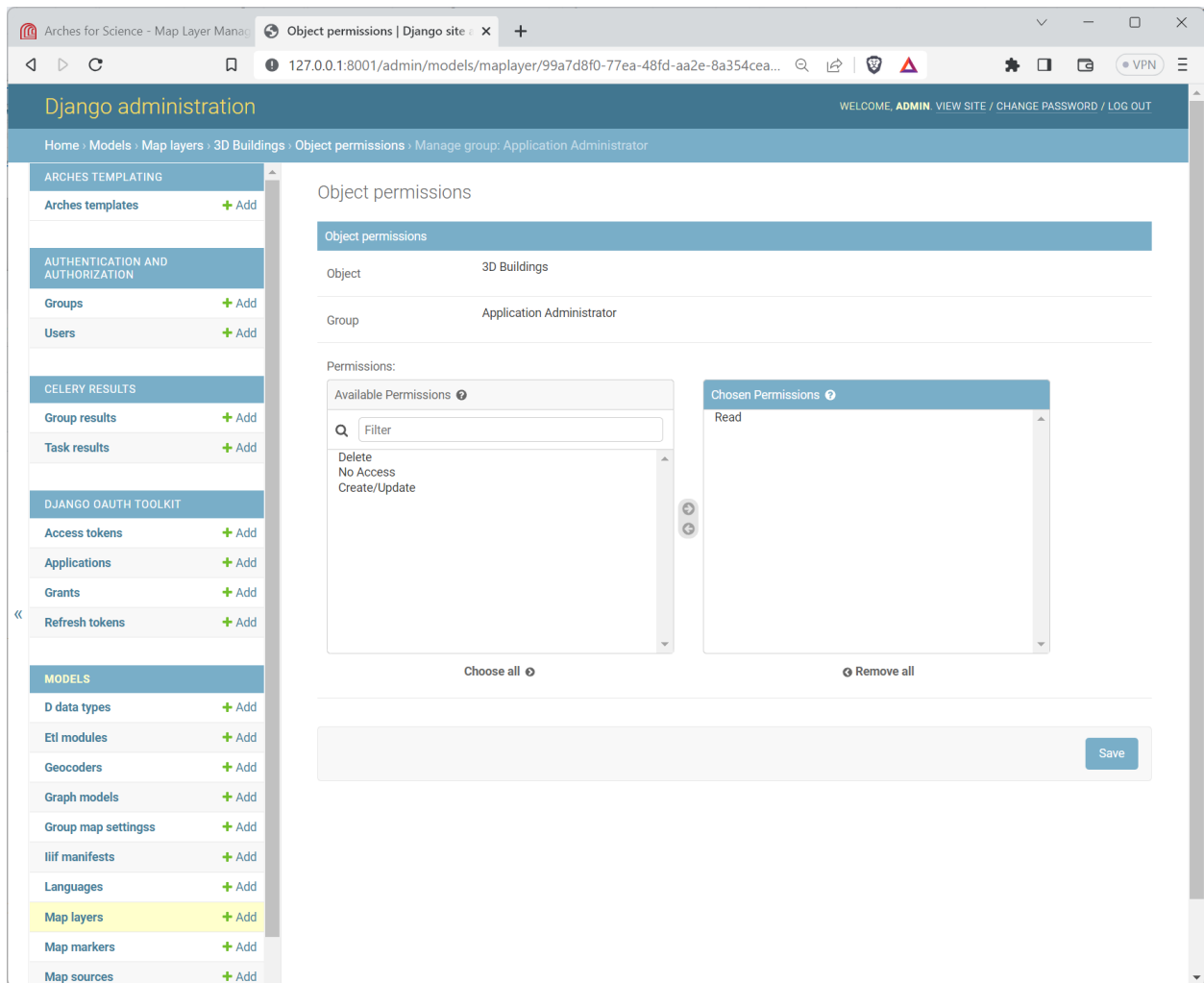


Fig. 23: Editing a group's specific permissions to a Map Layer in the Django Admin panel.

The spatial views are only able to represent the data in a flattened state, meaning that the data in nested cards are flattened into a single comma separated attribute value, with the card sort order honoured. Therefore, it is important to consider how to attribute the views being created.

Spatial Views Model Schema

The Spatial View model schema is defined as follows:

Spatialviewid

Unique identifier for the spatial view.

Schema

The database schema that the spatial view belongs to. `public` is used by default but if another is used then it must have already been created in the database.

Slug

This is will be joined with the Schema to form the name of the spatial view. This value must follow slug format of only lower-case letters, numbers, and hyphens. It cannot start with a number.

Description

The text that is added as a comment on the spatial view in the database , which can be accessed as metadata for consuming clients where supported. `pg_featureserv` for example will present this as the layer description.

Geometrynodeid

The UUID of the geojson-feature-collection node that underpins the geometry of the spatial view.

Ismixedgeometrytype

Boolean value that indicates whether the geometry of the spatial view is a mix of different geometry types. This is ideal where the spatial view will be used by a vector tile service.

Default value is `false`.

Attributenodes

A JSON object that contains a list of attribute object defining the UUIDs of the nodes that comprise the attributes of the spatial view and a text description of that attribute for metadata.

Note: The name of the attributes are automatically generated from the node name using Postgresql a compliant format.

```
[
  {
    "nodeid": "77e8f28d-efdc-11eb-afe4-a87eeabdefba",
    "description": "construction_phase_type"
  }, {
    "nodeid": "676d47ff-9c1c-11ea-b07f-f875a44e0e11",
    "description": "asset_name"
  }, {
    "nodeid": "325a2f33-efe4-11eb-b0bb-a87eeabdefba",
    "description": "primary_reference_number"
  }, {
    "nodeid": "ba345577-b554-11ea-a9ee-f875a44e0e11",
    "description": "description"
  }, {
    "nodeid": "b2133e72-efdc-11eb-a68d-a87eeabdefba",
```

(continues on next page)

(continued from previous page)

```
        "description": "use_phase_period"
    }, {
        "nodeid": "b2133e6b-efdc-11eb-aa04-a87eeabdefba",
        "description": "functional_type"
    }, {
        "nodeid": "77e8f29d-efdc-11eb-b890-a87eeabdefba",
        "description": "cultural_period"
    }
]
```

nodeid

The UUID of the node that needs adding. This must be in the same model and the **Geometrynodeid**.

description

The text description of the attribute, which will be added as metadata.

Isactive

Boolean value that indicates whether the spatial view is available. When set to `false` the spatial view is removed from the database, but allows the definition to remain. Setting to `true` recreates the spatial view in the database.

Default is `true`.

Creating your first spatial view

Django Admin

1. Logging in as a superuser, navigate to `/admin/models/spatialview` and click the **ADD SPATIAL VIEW +** button.
2. Complete the spatial view details and click the **Save** button.

Spatialviewid:	27318c10-adc4-421c-9e93-9c007ceee035
Schema:	public
Slug:	heritageasset
Description:	Heritage Assets provide information on the Monuments within the HER.
Geometrynodeid:	Node object (87d3d7dc-f44f-11eb-bee9-a87eeabdefba) ▼
<input type="checkbox"/> Ismixedgeometrytypes	
Attributenodes:	<pre>[{"nodeid": "77e8f28d-efdc-11eb-afe4-a87eeabdefba", "description": "construction_phase_type"}, {"nodeid": "676d47ff-9c1c-11ea-b07f-f875a44e0e11", "description": "asset_name"}, {"nodeid": "325a2f33-efe4-11eb-b0bb-a87eeabdefba", "description": "primary_reference_number"}, {"nodeid": "ba345577-b554-11ea-a9ee-f875a44e0e11", "description": "description"}, {"nodeid": "b2133e72-efdc-11eb-a68d-a87eeabdefba", "description": "use_phase_period"}, {"nodeid": "b2133e6b-efdc-11eb-aa04-a87eeabdefba"}]</pre>
<input checked="" type="checkbox"/> Isactive	
<div> Delete Save and add another Save and continue editing SAVE </div>	

SQL Insert

You can load the spatial view definition into the database using the following SQL:

```
INSERT INTO
public.spatial_views
(
    spatialviewid      ,
    schema             ,
    slug               ,
    description        ,
    ismixedgeometrytypes,
    attributenodes     ,
    isactive           ,
    geometrynodeid
)
VALUES
(
    '2a578e84-b21a-431d-8de0-59e4d46a88fb',
    'public',
    'artefact',
    'Defines information relating to the character of man made items of
↳ heritage significance as identified by the Portable Antiquities Scheme,
↳ includes individual artefacts, architectural items, artefact assemblages,
↳
```

(continues on next page)

(continued from previous page)

```

↪individual ecofacts and ecofact assemblages, and environmental samples.',
    false,
    [
      {
        "nodeid": "c30977b0-991e-11ea-ba04-f875a44e0e11",
        "description": "description"
      }, {
        "nodeid": "dd8032af-b494-11ea-8110-f875a44e0e11",
        "description": "primary_reference_number"
      }, {
        "nodeid": "dd8032b1-b494-11ea-a183-f875a44e0e11",
        "description": "legacy_id"
      }, {
        "nodeid": "99cfe72e-381d-11e8-882c-dca90488358a",
        "description": "from_date"
      }, {
        "nodeid": "22e7c550-afc2-11ea-a4a8-f875a44e0e11",
        "description": "repository_owner"
      }, {
        "nodeid": "50edbf22-ab25-11ea-a258-f875a44e0e11",
        "description": "storage_area_name"
      }, {
        "nodeid": "546b1630-3ba4-11eb-9030-f875a44e0e11",
        "description": "artefact_type"
      }, {
        "nodeid": "5b0dfb27-7fe2-11ea-8ac9-f875a44e0e11",
        "description": "artefact_name"
      }, {
        "nodeid": "99cff7f8-381d-11e8-a059-dca90488358a",
        "description": "to_date"
      }, {
        "nodeid": "99cfffdd1-381d-11e8-ab51-dca90488358a",
        "description": "cultural_period"
      }
    ]
    ,
    true,
    'f7ccc8b9-f447-11eb-9cb1-a87eeabdefba'
  );

```

Using the spatial views

To use the spatial views in your client application or datasource for a service, you will need to configure that client to connect to the database using the following credentials:

- *host*: the hostname of the arches database server
- *port*: the port of the arches database server
- *database*: the name of the arches database
- *user*: arches_spatial_views
- *password*: arches_spatial_views

If you are using a client that requires views to geometry type specific (for example ArcGIS), ensure that you have set `Ismixedgeometrytype` to false.

Important: Currently it is not possible to use the user/groups permissions to restrict access. You will need to manually create specific database users and assign them to the spatial views.

Example Usage

`pg_featureserv` and `pg_tileserv` are lightweight open source feature and vector tile service providers that can be used with these spatial views.

https://access.crunchydata.com/documentation/pg_featureserv/latest/ https://access.crunchydata.com/documentation/pg_tileserv/latest/

Once you have installed the application to run on your machine, open the config file located at:

`/path/to/pg_featureserv/config/pg_featureserv.toml`

Set the `DbConnection` setting to the following and restart the application:

```
DbConnection = "postgresql://arches_spatial_views:arches_spatial_views@<HOSTNAME>:<PORT>/  
-><DBNAME>"
```

User Guide

This User Guide should help Arches users perform basic data entry and retrieval tasks.

Creating and Editing 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** for it to appear in the Resource Manager.

Resource Editor

The Resource Editor is used to create new or edit existing Resources. On the left-hand side of the page you will see this Resource's "card tree", which shows all of the data entry cards that you can edit. Think of "creating data" as "adding cards".

To begin, select a card, enter data, and click Add. Some cards may allow multiple instances, in which case you will be able to add as many of the same type as you want.

Once you have saved data for a resource, you can see a full summary by selecting the top card. This is the resource report.

In some cases, cards will be nested within other cards, as in the example of adding a geo-location below.

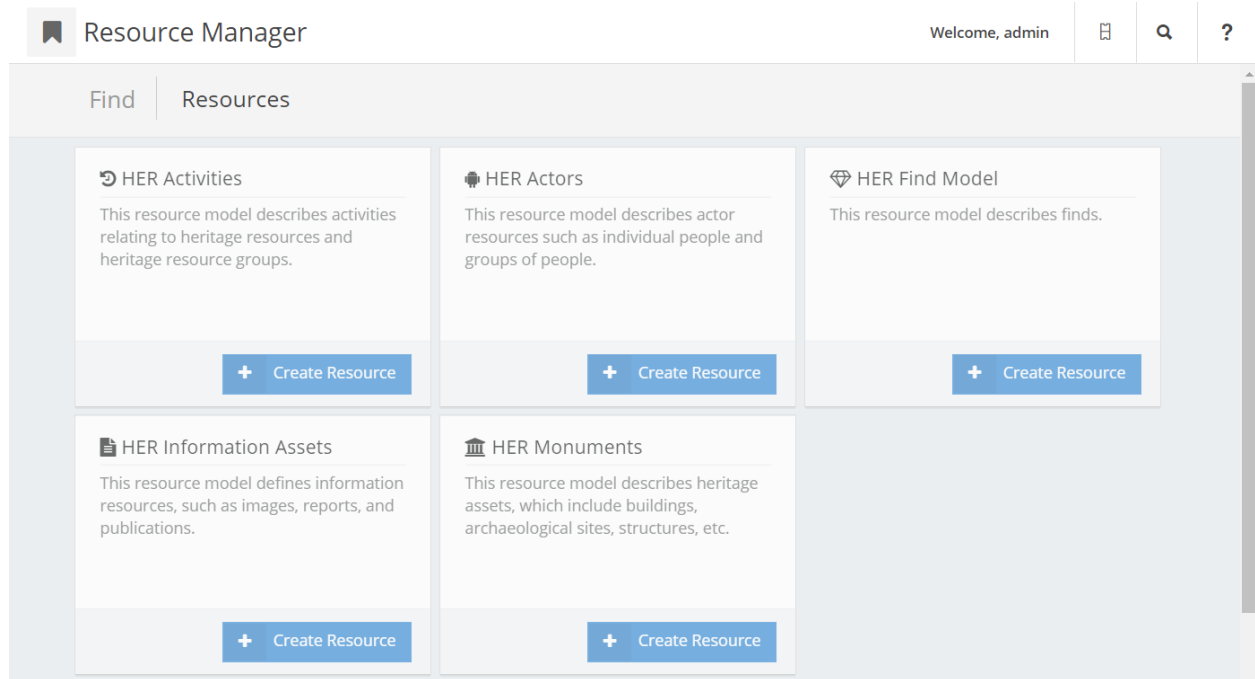


Fig. 24: Your Resource Manager page may look different than this image, depending on what Resource Models you have set up in your database.

Fig. 25: Simple data entry in Arches.

Fig. 26: Created nested data in Arches.

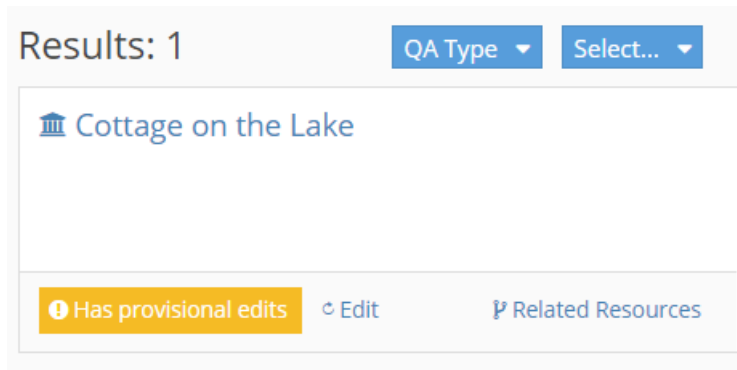
Provisional Edits

If you are a member of the Resource Editor group, all of your edits—either creating new resources or editing existing ones—will be considered “Provisional”. A member of the Resource Reviewer group can then approve your edits, making them “Authoritative”.

1. Resource Editor makes an edit:

✔ Your changes have been submitted for review

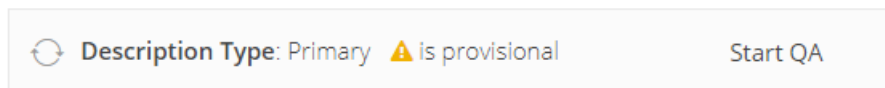
2. For Resource Reviewers, search results indicate provisional data:



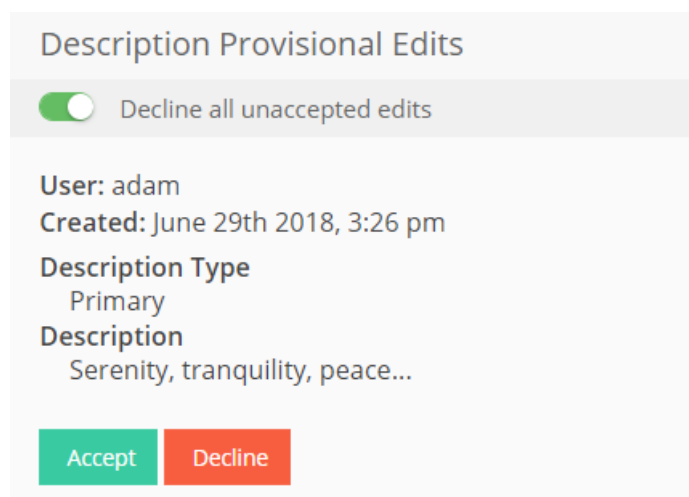
Resource Editors only see provisional data while using the resource editor.

3. Resource Reviewer will be prompted to Q/A the edit:

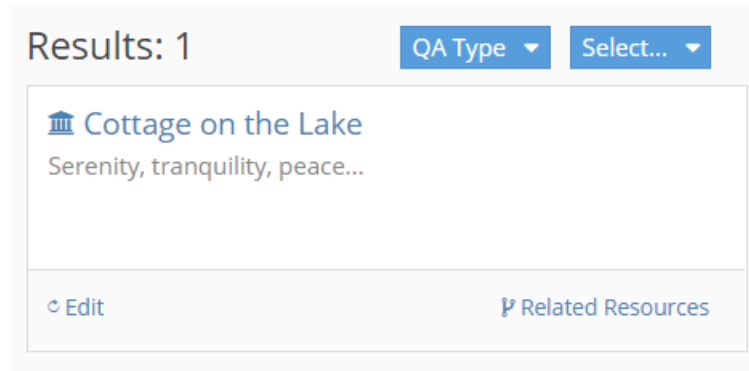
Saved Description



4. Accept or Decline:



5. Approved edits are immediately visible:



Tip: A Resource Reviewer can also use the “Q/A Type” search filter (see images above) to only find resources with (or without) provisional edits.

Related Resources

Warning: Managing generic relationships as described below is still an available feature in Arches. However, **this feature will soon be deprecated** in a future release. Users are strongly encouraged to use the *resource-instance* datatype to manage relationships between resource instances. The ability to visualize connections across *resource-instance* datatype nodes will accompany the deprecation of the generic resource relationship.

From the Resource Editor you can also access the Related Resources Editor, which is used to create a relationship between this resource and another in your database. To do so, open the editor, find the resource, and click Add. Your Resource Model will need to be configured to allow relations with the target Resource Model. If relations are not allowed, resources in the dropdown menu will not be selectable.

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.

Fig. 27: Creating a relationship between two resource in Arches, and adding properties to that relationship.

Note: Creating a relationship between two resources using the related resource editor is fundamentally different from creating a resource instance node in graph. Creating a relationship is good for making a visual “web” of resource relationships. Using a resource instance node in a Resource Model’s graph allows you to “embed” one resource inside of another.

Deleting Resources

Arches provides user interface features to delete node instances used to describe resource instances. Arches also provides user interface features to delete any single given resource instance. Finally, Arches provides a feature to delete all resource instances for a given resource model.

Select a Resource Instance to Edit or Delete

To find a resource instance that you want to delete, you can start by using the search interface (learn more here: [Searching](#)). Assuming you are logged in as a user with permissions to edit the resource instance of interest, you should find a link to **Edit** the resource instance in the search results. The animation below illustrates the use of search and the **Edit** link:

Delete a Node Instance for a Resource

Once you have opened a resource instance to edit, you have the option to delete (and then update if you choose) node instances associated with the resource instance. Node instance data are data that describe a given resource instances according to the structures defined by “branches”. The animation below illustrates different options one can use to delete an example annotation node instance:

Delete a Resource Instance Entirely

Sometimes you may wish to entirely delete a given resource instance. To do so, follow the directions above to find the resource instance you wish to delete and follow the **Edit** link. If you have edit permissions, on the resource instance edit page, you should see a **Manage** button toward the upper left corner of the page. Click on this, and select the **Delete Resource** option and then confirm your choice if you are sure you want to permanently delete the resource instance. See the animation below for an example resource instance deletion:

Delete ALL Resource Instances for a Resource Model

Arches also provides a user interface feature for bulk deletion of all resource instances created for a given resource model. To do so, navigate to the **Arches Designer** and select the **Resource Models** option. Then hover over the resource model of interest, and click on the **Manage** options on the right. You can then select the **Delete Instances** option, and after confirming your choice, you will delete *all* resource instances for that resource model. The animation below illustrates deletion of all resource instances for a resource model:

Warning: Obviously, deleting *all* resource instances for a given resource model can be a drastic measure. It may be a good idea to export and backup your data prior to such major changes (see [Export Commands](#)).

Searching

Help within the Arches application

The Arches application itself comes bundled with brief guides on the use of search features. Users can open these guides by clicking on the ? button in the top right of the search interface. The animation below illustrates activation of the search guide.

Term search and negation

Arches supports powerful features to search text. The text search can be used to match strings of text in any branch describing resource instances. One can also click on a search term for opposite effect, that is, to negate or exclude resource instances that match a given term. The animation below illustrates a text search to find resource instances containing the text “Iran” then a text search that excludes resource instances that contain the term “Iran”.

Search operators

The Arches search feature supports a number of operators that you can use to find and retrieve information. The following lists different operator types and their expected search behavior:

Exact: Putting your search term within quotes executes an exact search. An exact search only finds values that match then entire search string exactly, including case. For example, if the search term is “Excavation Unit” it will find the value “Excavation Unit 4”. It will *NOT* find “... this excavation unit ...”.

Like: This is a prefix based search. It will search for strings that begin with each term in the search string in any order and that there is a match for each term. It is not a “contains” search. For example, if the search term is “st bo” it will find “... stock book ...”, or “... books in stock ...”. It will *NOT* find a value with just “book” or find the value “last book” (where “last” ends in “st”).

Equals: This is a complete phrase search. It will only match full words in the exact order given. For example, if the search term is “stock book” it will find the value “Knoedler Stock Book 4”. It will *NOT* find “... books in stock ...”, or “... stocking new books ...”

Wildcard: This assumes the user is going to supply a search phrase exactly as they want. For example, a search for “st?ck” will find the values of “stock”, or “Stick”. It will *NOT* find “The car is stuck in the woods”, because the “stuck” is surrounded by other words. If you need a “contains-like” query, then you need to supply leading and trailing asterisks. So, “*st?ck*” *WILL* find “The car is stuck in the woods”.

In all the examples above we quote the search string for clarity. In the Arches application you wouldn’t need to put your search terms inside quotes.

Escaping search operator characters

Arches version 7.4 introduced features that allow users to use backslashes (“\”) to escape special characters used as search operators. For example, if one wanted to do a search for the string “sheep?”, where the retrieved text needs to include the question-mark “?” character, one would need to escape the “?” character so that it is **NOT** interpreted as a **Wildcard** search operator (see above). The string “sheep\?” escapes the “?” character and would search for the string “sheep?”.

Advanced Search

Arches also has “Advanced Search” options that enable users to search with specific branches to find resource instances. This adds much more precision to a search. For example, if you have resource instances described by both a “Color” branch and a “Material” branch a simple search for the term “gold” may return some unwanted results. The Advanced Search allows you to specify that you want to search within the “Material” branch, so a search for the term “gold” will return resource instances described by the material “gold”, not just the color “gold”.

Advanced Search is very powerful because you can use it to combine multiple search criteria together and compose complex queries. The animation below illustrates use of the Advanced Search option to search within specific branches.

1.1.3 For Developers and Software Customization

This section provides information on command line utilities, extensions, software and template customizations, API integrations, data modeling and other topics that are relevant to developers.

Developing with Arches

Arches is very flexible and customizable. This section provides guidance on how to use APIs to integrate Arches with other information systems, enhance accessibility, build custom extensions, and modify Arches to deploy custom features beyond the capabilities of standard (uncustomized, core) Arches.

If you are considering software development to customize Arches, please read the [Arches Customization Considerations](#) for an introduction about good practices to help make customizations easier to develop, sustain, and maintain.

Arches Customization Considerations

If you are leading a project or organization considering customizing Arches software, please read this document carefully. Customization is an inherently risky endeavor, especially if you need to maintain and support your information systems over multiple years.

The practices described here will reduce costs, reduce long term maintenance and security risks, and will lead to greater impact, enhanced sustainability, and open doors for future opportunities. That said, to maximize sustainability, security, maintainability, quality and impact, it is best practice to coordinate and discuss customization plans with the wider Arches open source community. If you haven’t already done so, please join the [Arches Community Forum](#)!

More Sustainable Pathways toward Customization

To increase the likelihood that customizations will have long term compatibility and maintainability with Arches, please use the customization patterns supported and documented by Arches. These patterns include:

- **Extensions** (see: [Creating Extensions](#))
 - card components
 - datatypes
 - functions
 - plugins
 - reports

- search filters
 - widgets
 - workflows
- **New map layers** (see: *Creating New Map Layers*)
 - **HTML export templates** (see: *Creating HTML Export Templates*)

Adherence to the extension design patterns helps to isolate your customizations from changes to the core of Arches. Following Arches extensions design patterns will also increase the likelihood that there will be relevant documentation and community help if the extensions need updates in the future. Certain customizations are easier to maintain over time. For example, an overwritten HTML template is generally simpler to upgrade than an inherited Arches Python class or an overwritten Django view. You should factor such considerations into long term resource planning.

Customizations Beyond Extensions

While the Arches extensions architecture offers a great deal of flexibility, there may be scenarios where you need additional flexibility. From a sustainability and maintenance perspective, this scenario has important risks that need to be understood and factored into long term resource planning and engineering.

Managing long term sustainability and maintenance risks should be a core software engineering focus. As much as possible, you should ideally isolate your customized module as much as possible from the core of Arches. One way preferred way to accomplish this is to develop **Arches Apps** (see: *Creating Applications*), which are discrete Python packages that can be integrated into one or more Arches projects. The **Arches Apps** documentation details their sustainability advantages.

API Based Customizations

The Arches *API* can be used to support customizations, especially those involving integration of Arches with other information systems. Channeling all connections between Arches and other systems through the API aligns with a design practice often described as “*Loose Coupling*”. By carefully limiting and simplifying how core Arches interfaces with external information systems, you reduce future maintenance burdens, because problems can be identified and fixed in a more focused manner.

Strive for Graceful Degradation

Things break over time, especially if they are customized and not widely supported by broader community. One way to help manage long term risks is to plan for “graceful degradation”. If your custom module is isolated from core Arches (via **Arches App** development and/or loose coupling of API integrations), then if it breaks or can no longer be maintained, the core Arches system should still be perfectly serviceable. Planning for obsolescence and the retirement of hard-to-maintain components is often essential in contexts where Arches is deployed, especially in the cultural heritage sector.

Creating a Development Environment

The following is our recommendation for creating an Arches environment that works well for developers. The first thing to consider is the general structure that will be in place, presumably all in the same directory:

Runtime Content

- ENV/ - A Python 3.8+ virtual environment (you can name this whatever you want).
- arches/ - The local clone of your fork of the [archesproject/arches](#) repo, this part of the code is often referred to as “core Arches.”
- my_project/ - The location of your Arches project. This is the app in which you will be making the majority of your front-end customizations (new images, new template contents, etc.).

Database Configuration Storage

- my_package/ - The location of your Arches package. Packages can store custom database definitions that you will create, and are loaded into a project through a one-time command line operation.

Setting Everything Up

Core Arches

1. Install all *software dependencies*, as well as [Git](#).

Note: You may also be planning to use externally hosted components, like a remote Postgres/PostGIS or Elasticsearch installation. In that case make sure you have the connection information handy, you will need it in a later step.

2. *Create a new Python 3.8+ virtual environment.*
3. Clone the core Arches repo

We recommend that you clone **your own fork** of the repo, but you can also clone [archesproject/arches](#) if you don't plan to contribute code.

```
(ENV)$ git clone https://github.com/archesproject/arches
(ENV)$ cd arches
```

4. Switch to the desired branch

You can switch between versions of core Arches by changing to whichever branch you want. For example:

```
(ENV)arches/$ git fetch
(ENV)arches/$ git checkout stable/6.0.0
```

will give you the stable branch for the 6.0.0 release.

5. Install the local core Arches

This is **instead of** using `pip install arches` which would install the pypi Arches distribution directly into ENV. When you install the local clone as shown below, any code changes you make inside of arches/ (like checking out a new git branch) will be immediately reflected in your runtime environment.

```
(ENV)arches/$ pip install -e .
(ENV)arches/$ pip install -r arches/install/requirements.txt
(ENV)arches/$ pip install -r arches/install/requirements_dev.txt
(ENV)arches/$ cd ..
```

Note: If you later switch to a new git branch, you may need to rerun `pip install -r arches/install/requirements.txt`, as the Python dependencies do change over the course of Arches releases.

The Project

You can now head to [Creating a New Arches Project](#) to proceed through the project creation and database setup steps. Additionally, we recommend that you turn the new project into a git repo, which aids development and deployment. Keep in mind:

- A `.gitignore` file will already be generated in your project.
- Make sure all sensitive information (db credentials, API keys, etc.) is stored in `settings_local.py`, **not** `settings.py`.

The Package (optional)

Think of the packages as external storage for complex database configurations like Resource Models, or custom components like Datatypes. A package allows you to back up and share this type of content outside of the project itself. In some cases, however, projects and packages can become interdependent.

Look at [Understanding Packages](#) for more information on how to create and maintain packages.

Overwriting Core Arches Content

In your project you can overwrite core Arches functionality in many ways. In general, doing so is preferable to directly altering any code in core Arches.

CSS (basic)

To overwrite existing (or add your own) style rules, create `project.css` in your project's media directory like this: `my_project/my_project/media/css/project.css` and place style content in there. By default, these rules are linked in the base Arches UI templates. To use these same rules on the splash page, add

```
<link href="{% static 'css/project.css' %}" rel="stylesheet">
```

to the bottom of the `<head>` tag in `my_project/my_project/templates/index.htm`.

Templates (.htm) and JS (.js) (intermediate)

For static files such as these, if you create a file in your project that matches the relative directory structure and name of that same file in core Arches, Django will inherit your new file and ignore the original Arches one.

Note: To add new Javascript libraries to your project, see [Adding JavaScript Dependencies](#).

Dynamic Content (advanced)

It is much more complex to override dynamic content like a core Arches **view**, but entirely possible. For example, you could create `views.py` in your project and define a new view class in it like this, which inherits a core Arches view class.

```
from arches.app.views.user import UserManagerView

class MyUserManagerView(UserManagerView):
    ## add a random print statement to make sure this class is used
    print("in MyUserManagerView")
    pass
```

and then in your `urls.py`, change

```
urlpatterns = [
    path("", include("arches.urls")),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

to

```
from .views import MyUserManagerView

urlpatterns = [
    # match and return your custom view before the default Arches url can get matched.
    path("user/", MyUserManagerView.as_view(), name="user_profile_manager"),
    path("", include("arches.urls")),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

which will cause `/user` to match your new view before the core Arches `/user` url is found. Thus, going to `localhost:8000/user` will still return the default Arches profile manager page, but it has been passed through your class. You can now add a `get()` method to your class and it will be called to return the view instead of `arches.app.views.user.UserManagerView().get()`.

Note: Remember: Arches is built with Django, so your best resource for more in-depth customization of projects is the [Django documentation](#) itself.

Warning: As a rule of thumb, the more complex the customizations are that you add to a project, the more difficult it will be to retain these changes when you upgrade to later core Arches versions.

Handling Upgrades

With the local clone of core Arches linked to your virtual environment, you can upgrade by simply pulling the changes to your local clone of the repo, or switching to a new release branch.

To upgrade projects, check the [release notes](#) which typically contain detailed instructions.

In general, you should always expect to

- 1) Reinstall Python dependencies in core Arches:

```
(ENV)$ cd arches
(ENV)arches/$ pip install -r requirements.txt
```

- 2) Apply database migrations in my_project:

```
(ENV)$ cd my_project
(ENV)my_project/$ python manage.py migrate
```

- 3) Reinstall javascript dependencies in my_project/my_project:

```
(ENV)$ cd my_project/my_project
(ENV)my_project/my_project$ yarn install
```

Finally, if you have added custom logic or content to your project, you must make sure to account for any changes in the core Arches content that you have overwritten or inherited.

Running Tests

Tests must be run from core Arches. Enter arches/ and then use:

```
(ENV)arches/$ python manage.py test tests --pattern="*.py" --settings="tests.test_
↪ settings"
```

It is possible that you will need to add or update settings_local.py inside of arches/ in order for the tests to connect to Postgres and Elasticsearch.

Arches and Elasticsearch

Arches uses [Elasticsearch](#) as its search engine. A handful of settings.py variables point your Arches project to an Elasticsearch installation, in which your indexes will be created. An ELASTICSEARCH_PREFIX string is prepended to all of your project's indexes, meaning that a single Elasticsearch installation can be used by multiple projects.

One important thing to remember: **Elasticsearch indexes are replicable derivatives of your Arches database**, meaning that they can safely be dropped and recreated at any time. Similarly, if you need to change or upgrade your Elasticsearch setup, you need only update some settings and then reindex your database.

You can install Elasticsearch locally alongside Arches—read on for how to do that. You can also use managed Elasticsearch solutions by cloud providers like [AWS](#).

Installing Elasticsearch

The easiest way to install Elasticsearch is to download and unpack their archived releases. Archives are available at <https://www.elastic.co/downloads/past-releases/elasticsearch-{release number}>, e.g. <https://www.elastic.co/downloads/past-releases/elasticsearch-8-5-1>.

Download the release for your OS and architecture and then unpack/unzip it. For example, installing 8.5.1 on Ubuntu Linux looks like:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-8.5.1-linux-x86_
↳64.tar.gz
tar -zxvf elasticsearch-8.5.1-linux-x86_64.tar.gz
```

A full installation is now in `./elasticsearch-8.5.1`, which you can start by running `./elasticsearch-8.5.1/bin/elasticsearch` (see below).

On Windows you will need the Windows release which is a ZIP archive, but the process is basically the same.

Development Configuration

Elasticsearch 8 introduced new security features. While you are working with Arches locally, i.e. during development, you can safely disable these features. *Do not disable security features in production.*

Make two changes:

1. In your config file, find `xpack.security.enabled = true` and set it to `xpack.security.enabled = false`. Now start/restart Elasticsearch (see below).

The config file is typically found at `{path-to-elasticsearch}\config\elasticsearch.yml`. If you installed the Debian package, you'll find it at `/etc/elasticsearch/elasticsearch.yml`.

2. In your Arches project's `settings.py` or `settings_local.py`, add

```
ELASTICSEARCH_HOSTS = [{'scheme': 'http', 'host': 'localhost', 'port': 9
↳2000}
↳ELASTICSEARCH_HTTP_PORT}]
```

This overwrites the default `ELASTICSEARCH_HOSTS` variable, which has the scheme set to `https`.

Running Elasticsearch

Linux/macOS:

After unpacking the archive, use

```
{path-to-elasticsearch}/bin/elasticsearch
```

To run in the background, add `-d` to that command. To stop the background process, use `ps aux | grep elasticsearch` to get the process id, and then `sudo kill <process id> -9`.

Windows:

On Windows, double-click the `{path-to-elasticsearch}\bin\elasticsearch.bat` batch file to run the process in a new console window.

To make sure Elasticsearch is running correctly, use

```
curl localhost:9200
```

You should get a JSON response that includes “You Know, For Search...”. You can also use the Chrome plugin [ElasticSearch Head](#) to view your instance in a browser at `localhost:9200`.

For more information, please visit the official [Elasticsearch documentation](#).

Important:

1. By default, Elasticsearch uses 2GB of memory (RAM). For basic development purposes, we have found it to run well enough on 1GB. Use `ES_JAVA_OPTS="-Xms1g -Xmx1g" ./bin/elasticsearch -d` to set the memory allotment on startup ([read more](#)). You can use the same command to give **more** memory to Elasticsearch in a production setting.

Important: If you get an empty response from `curl localhost:9200`, this is likely because Elasticsearch security features are not probably set up, see [Development Configuration](#) above.

Using the Kibana Dashboard

<https://github.com/archesproject/arches-docs/issues/217>

Reindexing The Database

You may need to reindex the entire database now and then. This can be helpful if a bulk load failed halfway through, or if you need to point your database at a different Elasticsearch installation.

Be warned that this process can take a long time if you have a lot of resources in your database. Also, if you are in DEBUG mode it can cause your server to run out of memory.

See [reindex the database](#) for the commands needed for reindexing.

Using Multiple Nodes

In production it’s advisable to have multiple Elasticsearch instances working together as nodes of a single cluster. To do this, you need to install a second Elasticsearch instance, and change the `config/elasticsearch.yml` file in each instance. Note that the cluster and node names can be whatever you want, as long as the `cluster.name` is the same in both instances and the `node.name` is unique to each one.

Master (Original) Node Config

```
http.port: 9200

cluster.name: arches-app
node.name: arches-app-node1

node.master: true
node.data: true
```

Secondary Node Config

```

http.port: 9201

cluster.name: arches-app
node.name: arches-app-node2

node.master: false
node.data: true

```

Leave all other parameters untouched.

You'll need to start/stop each of these instances individually, but you should always have both running. When they are, the secondary node will automatically find the master node and the indices will be replicated between the two.

Nothing about your project's `settings.py` should change; Arches need only connect to the original Elasticsearch instance as before. However, you'll see now in the console output that the cluster health will be `[GREEN]` when you have two nodes running (it's `[YELLOW]` if you only have one).

See also:

Here's some [background](#) and a [stack overflow question](#) with instructions for adding a node.

Adding a Custom Index

Arches allows you to create a custom index of resource data for your specific use case (for use in Kibana for example).

To add a new custom index create a new python module and add to it a class that inherits from `arches.app.search.base_index.BaseIndex` and implements the `prepare_index` and `get_documents_to_index` methods.

Example custom index:

```

from arches.app.search.base_index import BaseIndex

class SampleIndex(BaseIndex):
    def prepare_index(self):
        self.index_metadata = {"mappings": {"_doc": {"properties": {"tile_count": {"type": "keyword"}, "graph_id": {"type": "keyword"}}}}}
        super(SampleIndex, self).prepare_index()

    def get_documents_to_index(self, resourceinstance, tiles):
        return ({"tile_count": len(tiles), "graph_id": resourceinstance.graph_id},
        str(resourceinstance.resourceinstanceid))

```

add this to your `settings_local.py` file

```

ELASTICSEARCH_CUSTOM_INDEXES = [{
    'module': '{path to file with SampleIndex class}.SampleIndex',
    'name': 'my_new_custom_index' <-- follow ES index naming rules, use this name to
    register in Elasticsearch
}]

```

Register your index in Elasticsearch: see [register a custom index](#)

Arches Use of the Django ORM

Arches is built on Django, a powerful, popular, well-supported and well-documented Python language web framework. This guide is intended to help guide developers already familiar with Django to better understand the Arches backend. The main focus here will center on how Arches uses the [Django Object Relational Model \(ORM\)](#) to power a highly configurable (and semantic, if one chooses to use ontologies) abstract [Data Model](#).

The Arches [Data Model](#) documentation provides an invaluable reference to understand Arches implementations of Django ORM models. This page provides more of a “guided tour” that illustrates how the Arches information you see in a browser may be reflected in Django [query sets](#) and objects (individual records).

Exploring a (Nearly) Empty Database

In this guide, we will start with a freshly installed and nearly empty Arches instance to make exploration easier. If you haven’t yet installed Arches, please review and follow this [Installing](#) guide. To avoid permissions complications, login to your new Arches instance as an administrator (super user). We will then use the [Arches Designer](#) to set up a simple Branch and a Resource Model.

Important:

- The UUIDs will be randomly generated and differ from these examples. To test this on your own Arches instance, you’ll need to replace UUID identifiers with those present in your own instance.

1. Build a Branch

Use the Arches Graph designer to make a branch and a resource model. In this demonstration case, we’re making a simple branch for “Name” with two child nodes (“Given Name” and “Surname”).

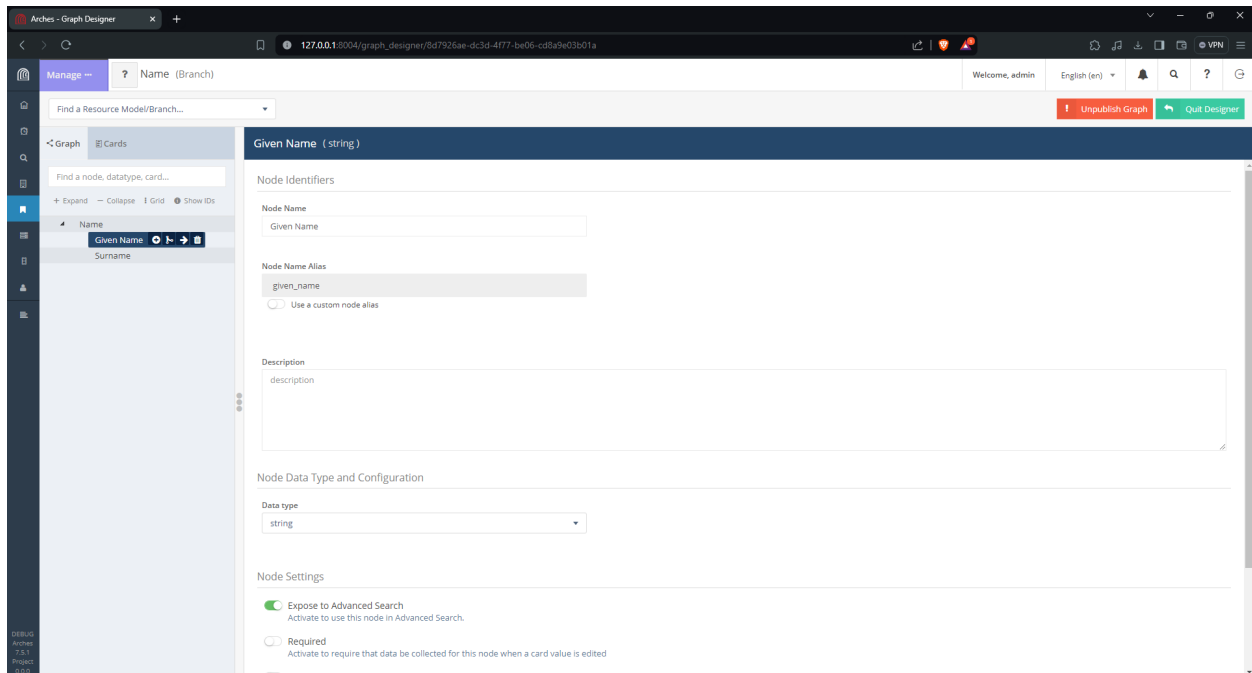


Fig. 28: Arches Designer user interface to create a new “Name” branch.

2. Build a Resource Model

After publishing this new “Name” branch, we can use it to describe resource models. Here, we’re in the process of adding the “Name” branch to a “Person” resource model.

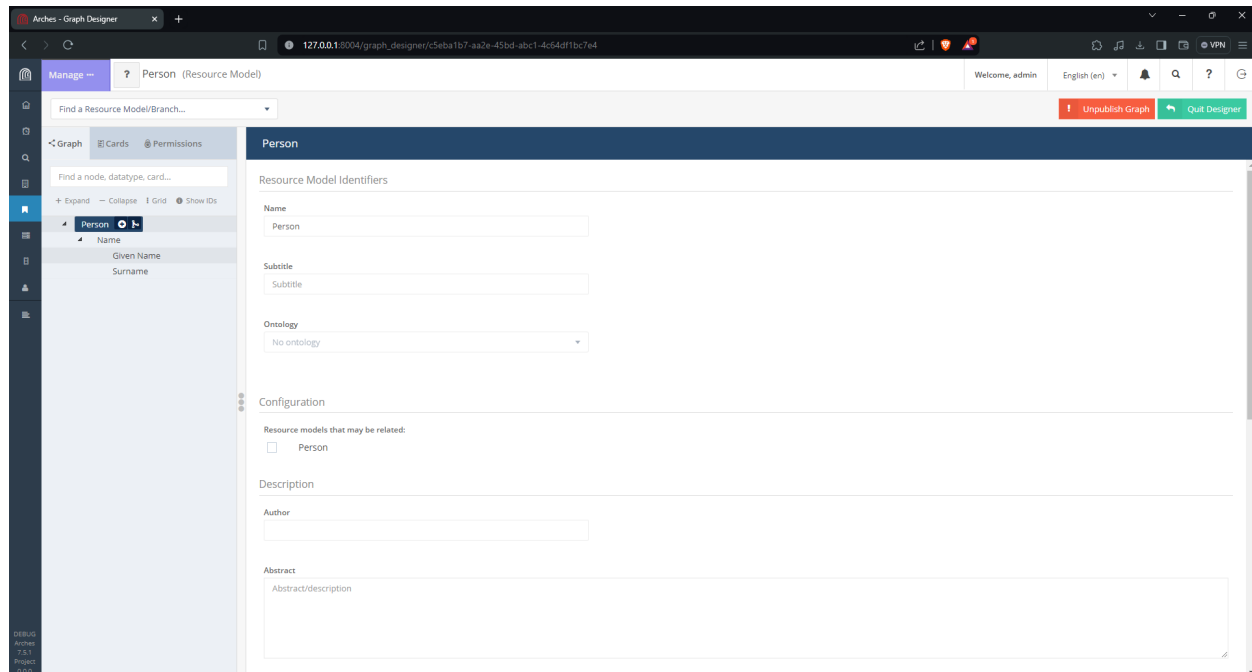


Fig. 29: Arches Designer user interface to create a new “Person” resource model.

This results in the “Person” resource model with a “Name” branch. After one clicks on the “Publish Graph” button, we can create business data. In our example, that business data will include resource instances (of the Person resource model) and names (configured with the Name branch).

3. Add a Resource Instance

Using the “Add New Resource” user interface, we can add a Person resource instance with name information. Once you save your new resource instance, let’s explore how the information is represented in the Django ORM used by Arches.

4. Open a Terminal to Explore the ORM

Now that you have used the Arches user interface to define a branch, a resource model, and have used these to create a resource instance, we can turn our attention to exploring how this information is represented in the Arches implementation of the Django ORM.

Assuming you’ve activated your virtual environment for Arches, use a terminal to open a shell into the Arches Django application:

```
python manage.py shell
```

Your terminal should display something like this:

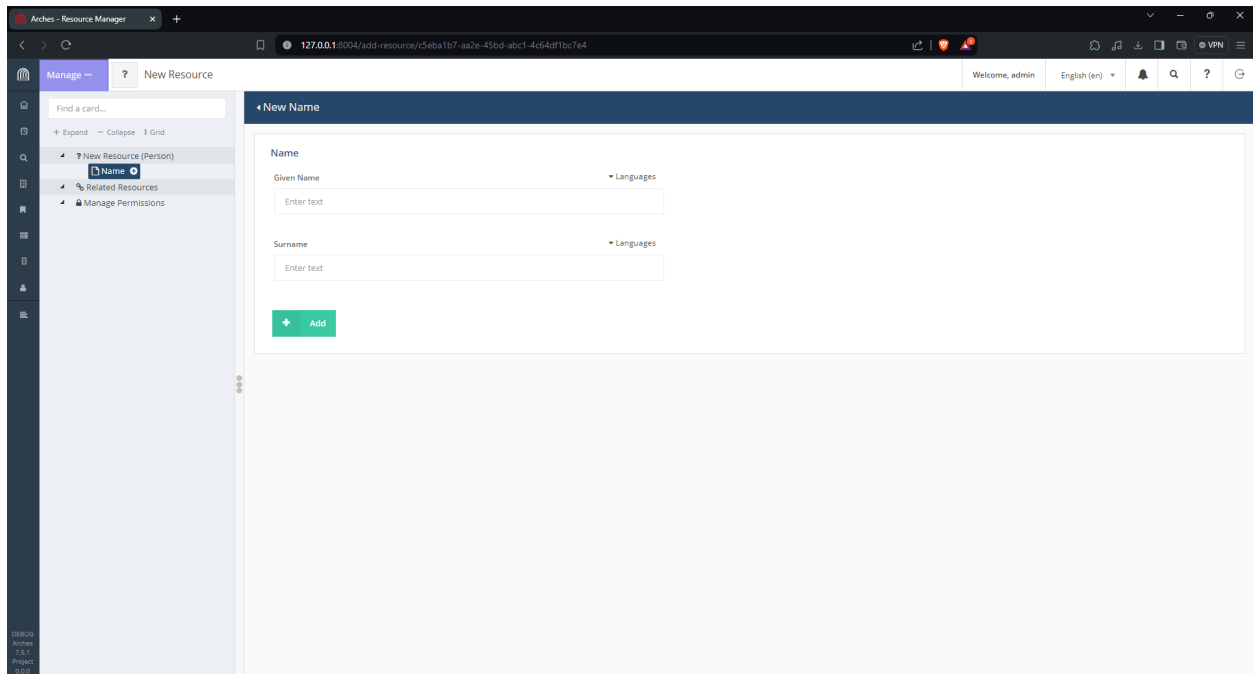


Fig. 30: Adding a resource instance

```
Python 3.11.8 (main, Mar 12 2024, 11:41:52) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

5. Import Arches Models and Explore the GraphModel

Now we should import some of the key Django models used by Arches to organize data. After importing these models, we can investigate how Arches represents the “Name” branch and the “Person” resource model that we already created using the user interface.

```
Python 3.11.8 (main, Mar 12 2024, 11:41:52) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from arches.app.models.models import GraphModel
```

Let’s first take a look at the GraphModel. The GraphModel is used to store records of both branches and resource models.

```
>>> gr_qs = GraphModel.objects.all()
>>> gr_qs.count()
3
```

You’ll see we have 3 objects in our queryset to select all items from the GraphModel. But we only made one branch, and one resource model! Where does the other GraphModel object come from?

To answer this question, let’s investigate further by looking at an individual object from the query set. The `vars()` method outputs the object as a dict, making it easier to see the information that it contains.

```
>>> gr_obj = gr_qs.last() # Get the last object in this queryset
>>> vars(gr_obj)
{'_state': <django.db.models.base.ModelState object at 0x7f40110d3350>,
'graphid': UUID('ff623370-fa12-11e6-b98b-6c4008b05c4c'),
'name': <arches.app.models.fields.i18n.I18n_String object at 0x7f40110d18d0>,
'description': <arches.app.models.fields.i18n.I18n_String object at 0x7f40110d2f50>,
'deploymentfile': None, 'author': ' ', 'deploymentdate': None, 'version': '', 'isresource':
↳ True,
'iconclass': 'fa fa-sliders', 'color': None,
'subtitle': <arches.app.models.fields.i18n.I18n_String object at 0x7f40110d3050>,
'ontology_id': None, 'jsonldcontext': None, 'template_id': UUID('50000000-0000-0000-0000-
↳ 000000000001'),
'config': {}, 'slug': None, 'publication_id': UUID('e437751c-e234-11ee-a712-0242ac120005
↳ ')}

```

That looks a little difficult to understand especially because the name attribute has an `I18n_String` object. The `I18n_String` object is used by Arches to support internationalization. One can see the `I18n_String` object with:

```
>>> gr_obj.name.value
{'en': 'Arches System Settings'}
>>> str(gr_obj.name)
'Arches System Settings'

```

As you can see, the value of the `I18n_String` object is a JSON formatted string. Language codes (“en” in this case) are used as keys to different multi-lingual strings. One can get the string value for an `I18n_String` object, in the default language, with `str()`. So to output a more legible overview of try `GraphModel` queryset, try:

```
>>> [(gr_obj.graphid, str(gr_obj.name), gr_obj.isresource) for gr_obj in gr_qs]
[(UUID('ff623370-fa12-11e6-b98b-6c4008b05c4c'), 'Arches System Settings', True),
(UUID('c5eba1b7-aa2e-45bd-abc1-4c64df1bc7e4'), 'Person', True),
(UUID('8d7926ae-dc3d-4f77-be06-cd8a9e03b01a'), 'Name', False)]

```

Now we have a more clear picture of what’s contained in the `GraphModel` queryset. The ‘Arches System Settings’ object was created in the process that set up the current Arches project. The two `GraphModel` objects that we created (‘Person’ and ‘Name’) are also present in the `GraphModel` queryset. The `isresource` attribute indicates that the ‘Person’ `GraphModel` object is a resource model. We can get an individual `GraphModel` object for our “Person” resource model by querying the Django ORM as so:

```
>>> person_resource_model_obj = GraphModel.objects.get(graphid='c5eba1b7-aa2e-45bd-abc1-
↳ 4c64df1bc7e4')
>>> str(person_resource_model_obj.name)
'Person'

```

6. Resource Instances and their GraphModels

Now that we’ve explored the `GraphModel` and `I18n_String` objects, let’s take a look at how Arches uses the Django ORM to manage “business data”. In the context of Arches, “business data” means the database records (resource instances and their descriptions) managed within an Arches instance. At this point, we’re assuming you have created a “Person” resource instance as discussed in Step 3 above. To start exploring business data, start with the following:

```
>>> from arches.app.models.models import ResourceInstance
>>> from arches.app.models.resource import Resource

```


In this case `Resource` is a proxy model (see [Django’s documentation for proxy models](#)) for `ResourceInstance`. The proxy model `Resource` adds some additional Python methods to the `ResourceInstance` model. Most of the discussion below will focus on use of the `Resource` proxy model. So let’s make a `Resource` queryset and inspect the first object within this queryset:

```
>>> r_qs = Resource.objects.all()
>>> r_obj = r_qs.first()
>>> vars(r_obj)
{'_state': <django.db.models.base.ModelState object at 0x7f400f5f3150>,
 'resourceinstanceid': UUID('a106c400-260c-11e7-a604-14109fd34195'),
 'graph_id': UUID('ff623370-fa12-11e6-b98b-6c4008b05c4c'),
 'graph_publication_id': UUID('f0a0bf6a-65af-46f2-9c08-62e21a56dffb'),
 'name': <arches.app.models.fields.i18n.I18n_String object at 0x7f400faa7010>,
 'descriptors': {'ar': {'name': None, 'map_popup': None, 'description': None},
 'en': {'name': None, 'map_popup': None, 'description': None},
 'he': {'name': None, 'map_popup': None, 'description': None}},
 'legacyid': 'a106c400-260c-11e7-a604-14109fd34195',
 'createdtime': datetime.datetime(2024, 3, 14, 13, 58, 48, 564559),
 'tiles': [], 'descriptor_function': None,
 'serialized_graph': None, 'node_datatypes': None}
```

We can see right away that this `Resource` object has a `graph_id` that matches the `graph_id` of the ‘Arches System Settings’ that we explored earlier. You can see this by following the related objects as below:

```
>>> str(r_obj.graph.name)
'Arches System Settings'
```

This particular resource instance that’s associated with the ‘Arches System Settings’ was also created in the process that set up the current Arches project. Let’s look for the resource instance from the “Person” model that we created. To do so, we can make a new `Resource` queryset filtering by resource instances that use the “Person” resource model.

```
>>> person_r_qs = Resource.objects.filter(graph=person_resource_model_obj)
>>> person_r_qs.count()
1
```

As expected, since we’ve only made 1 resource instance using the “Person” resource model the `person_r_qs` queryset has 1 object in it. Let’s take a look at this Person resource instance:

```
>>> person_r_obj = person_r_qs[0]
>>> str(person_r_obj.graph.name) # See the 'Person' Resource Model (GraphModel)
'Person'
>>> vars(person_r_obj)
{'_state': <django.db.models.base.ModelState object at 0x7fb15ef2ad50>,
 'resourceinstanceid': UUID('e9012e8c-f1cc-4ade-84ea-9b73ed8cccf9'),
 'graph_id': UUID('c5eba1b7-aa2e-45bd-abc1-4c64df1bc7e4'),
 'graph_publication_id': UUID('b338fef6-eba6-11ee-8bd0-0242ac120005'),
 'name': <arches.app.models.fields.i18n.I18n_String object at 0x7fb15ef29f90>,
 'descriptors': {'ar': {'name': None, 'map_popup': None, 'description': None},
 'en': {'name': None, 'map_popup': None, 'description': None},
 'he': {'name': None, 'map_popup': None, 'description': None}},
 'legacyid': None, 'createdtime': datetime.datetime(2024, 3, 26, 14, 46, 41, 394410),
 'tiles': [], 'descriptor_function': None,
 'serialized_graph': None, 'node_datatypes': None}
```

7. Resource Instances and their Description

In the example above, you’ll see that the “descriptors” attribute has a dictionary keyed by different language codes (in this case ‘ar’, ‘en’, and ‘he’). The descriptors attribute is used by Arches to populate information about resource instances in the user interface. In the example above, these descriptors have yet to be configured. Let’s see what happens when we do configure resource instance descriptors.

Use the Arches Graph designer and navigate to the Resource Models tab. Hover over the “Person” resource model until you see the “Manage” button, and select the “Manage Functions” option. You can then configure the “Display Name” to use the “Name” (Card) with two child nodes “Given Name” and “Surname” similar to below:

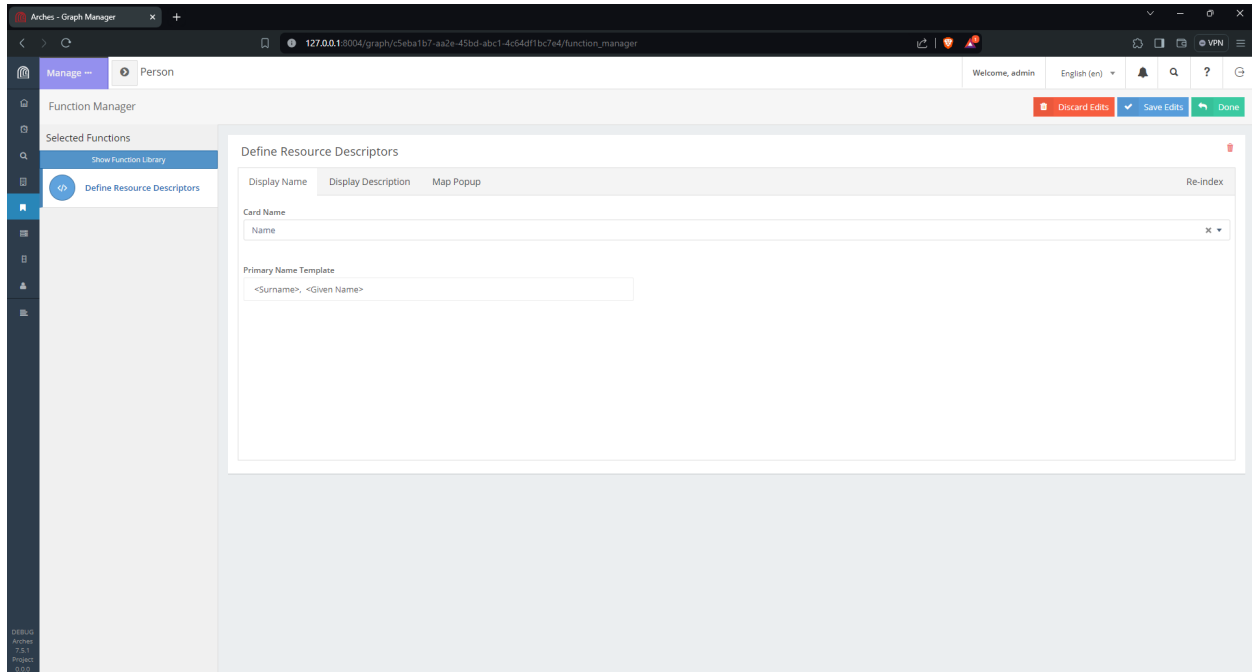


Fig. 31: Arches Designer to configure the Display Name for the “Person” resource model

Once you have finished this, click the “Re-index” action for your changes to take effect and so the changes become evident in the Arches search. Turning back to the terminal and the Python for Arches, we can see our changes on the name descriptor are reflected in the resource instance. The first thing is to make sure our resource instance object gets updated to reflect its current state in the database. Django model instance objects have a built in `refresh_from_db()` method to do this:

```
>>> person_r_obj.refresh_from_db()
>>> vars(person_r_obj)
{'_state': <django.db.models.base.ModelState object at 0x7f400ef5d490>,
 'resourceinstanceid': UUID('e9012e8c-f1cc-4ade-84ea-9b73ed8cccf9'),
 'graph_id': UUID('c5eba1b7-aa2e-45bd-abc1-4c64df1bc7e4'),
 'graph_publication_id': UUID('b338fef6-eba6-11ee-8bd0-0242ac120005'),
 'name': <arches.app.models.fields.i18n.I18n_String object at 0x7f400f5cf110>,
 'descriptors': {'ar': {'name': '', 'map_popup': None, 'description': None},
 'en': {'name': 'Summers, Buffy', 'map_popup': None, 'description': None},
 'he': {'name': '', 'map_popup': None, 'description': None}},
 'legacyid': None, 'createdtime': datetime.datetime(2024, 3, 26, 14, 46, 41, 394410),
 'tiles': [], 'descriptor_function': None,
 'serialized_graph': None, 'node_datatypes': None}
```

You can see that there's a change in the 'descriptors' attribute. This is still a little hard to read. Fortunately, the Resource proxy model has some useful functions that can help us understand this object. Here's an example of using a method that comes with the Resource proxy model. Note also the name attribute (a `I18n_String` object) will also return the same value:

```
>>> person_r_obj.displayname()
'Summers, Buffy'
>>> str(person_r_obj.name)
'Summers, Buffy'
```

Congratulations! You can now see how some Arches information configured and rendered in the browser is represented in the Django ORM used by Arches.

8. Resource Instances and their Tile Data

Let's continue this investigation by making a `TileModel` queryset filtered by the resource instance in our `Person` model. The following makes this query set display its count.

```
>>> from arches.app.models.models import TileModel
>>> t_qs = TileModel.objects.filter(resourceinstance=person_r_obj)
>>> t_qs.count()
1
```

We can then explore what the one `TileModel` object looks like when rendered as a Python dictionary. Doing so reveals how Arches represents a resource instance's descriptive attributes as "tile data".

```
>>> t_obj = t_qs[0]
>>> vars(t_obj)
{'_state': <django.db.models.base.ModelState object at 0x7f400ee3d150>,
'tileid': UUID('c7194a01-ab74-44dd-9c52-a12ded792fdc'),
'resourceinstance_id': UUID('e9012e8c-f1cc-4ade-84ea-9b73ed8cccf9'),
'parenttile_id': None,
'data': {'a9d08578-eba6-11ee-be3e-0242ac120005': {'ar': {'value': '', 'direction': 'rtl'}
↪,
'de': {'value': '', 'direction': 'ltr'}, 'el': {'value': '', 'direction': 'ltr'},
'en': {'value': 'Buffy', 'direction': 'ltr'}, 'fr': {'value': '', 'direction': 'ltr'},
'he': {'value': '', 'direction': 'rtl'}, 'pt': {'value': '', 'direction': 'ltr'},
'ru': {'value': '', 'direction': 'ltr'}, 'zh': {'value': '', 'direction': 'ltr'},
'en-US': {'value': '', 'direction': 'ltr'}},
'a9d08604-eba6-11ee-be3e-0242ac120005': {'ar': {'value': '', 'direction': 'rtl'},
'de': {'value': '', 'direction': 'ltr'}, 'el': {'value': '', 'direction': 'ltr'},
'en': {'value': 'Summers', 'direction': 'ltr'}, 'fr': {'value': '', 'direction': 'ltr'},
'he': {'value': '', 'direction': 'rtl'}, 'pt': {'value': '', 'direction': 'ltr'},
'ru': {'value': '', 'direction': 'ltr'}, 'zh': {'value': '', 'direction': 'ltr'},
'en-US': {'value': '', 'direction': 'ltr'}}},
'nodegroup_id': UUID('a9d083d4-eba6-11ee-be3e-0242ac120005'),
'sortorder': 0, 'provisionaledits': None}
```

The data attribute of this `TileModel` object has a dictionary with a nested structure keyed first by `nodeid` and then by language codes (see [comments in the source code here](#)). To learn more about how the `nodeid` is used in representing graphs of data, please review [Graph Definition](#).

One can also get tile data for an instance of the Resource proxy model by calling the `load_tiles()` method that's defined for that proxy model. This will populate a list (not a queryset) of `Tile` objects for a given resource instance.

```
>>> person_r_obj.load_tiles()
>>> person_r_obj.tiles
[<TileModel: TileModel object (c7194a01-ab74-44dd-9c52-a12ded792fdc)>]
>>> vars(person_r_obj.tiles[0])
{'_state': <django.db.models.base.ModelState object at 0x7f400ee3d150>,
'tileid': UUID('c7194a01-ab74-44dd-9c52-a12ded792fdc'),
'resourceinstance_id': UUID('e9012e8c-f1cc-4ade-84ea-9b73ed8cccf9'),
'parenttile_id': None,
'data': {'a9d08578-eba6-11ee-be3e-0242ac120005': {'ar': {'value': '', 'direction': 'rtl'}
↪,
'de': {'value': '', 'direction': 'ltr'}, 'el': {'value': '', 'direction': 'ltr'},
'en': {'value': 'Buffy', 'direction': 'ltr'}, 'fr': {'value': '', 'direction': 'ltr'},
'he': {'value': '', 'direction': 'rtl'}, 'pt': {'value': '', 'direction': 'ltr'},
'ru': {'value': '', 'direction': 'ltr'}, 'zh': {'value': '', 'direction': 'ltr'},
'en-US': {'value': '', 'direction': 'ltr'}},
'a9d08604-eba6-11ee-be3e-0242ac120005': {'ar': {'value': '', 'direction': 'rtl'},
'de': {'value': '', 'direction': 'ltr'}, 'el': {'value': '', 'direction': 'ltr'},
'en': {'value': 'Summers', 'direction': 'ltr'}, 'fr': {'value': '', 'direction': 'ltr'},
'he': {'value': '', 'direction': 'rtl'}, 'pt': {'value': '', 'direction': 'ltr'},
'ru': {'value': '', 'direction': 'ltr'}, 'zh': {'value': '', 'direction': 'ltr'},
'en-US': {'value': '', 'direction': 'ltr'}}},
'nodegroup_id': UUID('a9d083d4-eba6-11ee-be3e-0242ac120005'),
'sortorder': 0, 'provisionaledits': None}
```

9. Concluding the Tour

As shown above, Arches uses the Django ORM to represent data using python models. In order to gain mastery over Arches data modeling, there are more implementation details to understand, and these will be further described in future updates to this documentation.

API

General Notes

Arches allows any parameters to be passed in via custom HTTP headers OR via the querystring. All requests to secure services require users to pass a “Bearer” token in the authentication header

To use a an HTTP header to pass in a parameter use the form:

```
HTTP-X-ARCHES-{upper case parameter name}.
```

So, for example, these are equivalent requests

```
curl -H "X-ARCHES-FORMAT: json-ld" http://localhost:8000/mobileprojects
curl http://localhost:8000/mobileprojects?format=json-ld
```

If both a custom header and querystring with the same name are provided, then the querystring parameter takes precedence.

In the following example “html” will be used as the value for the “format” parameter.

```
curl -H "X-ARCHES-FORMAT: json-ld" http://localhost:8000/mobileprojects?  
↪format=html
```

Note: Querystring parameters are case sensitive. Behind the scenes, custom header parameters are converted to lower case querystring parameters.

In the following example there are 3 different parameters (“format”, “FORMAT”, and “Format”) with 3 different values (“html”, “json”, and “xml”) respectively

```
http://localhost:8000/mobileprojects?format=html&FORMAT=json&Format=xml
```

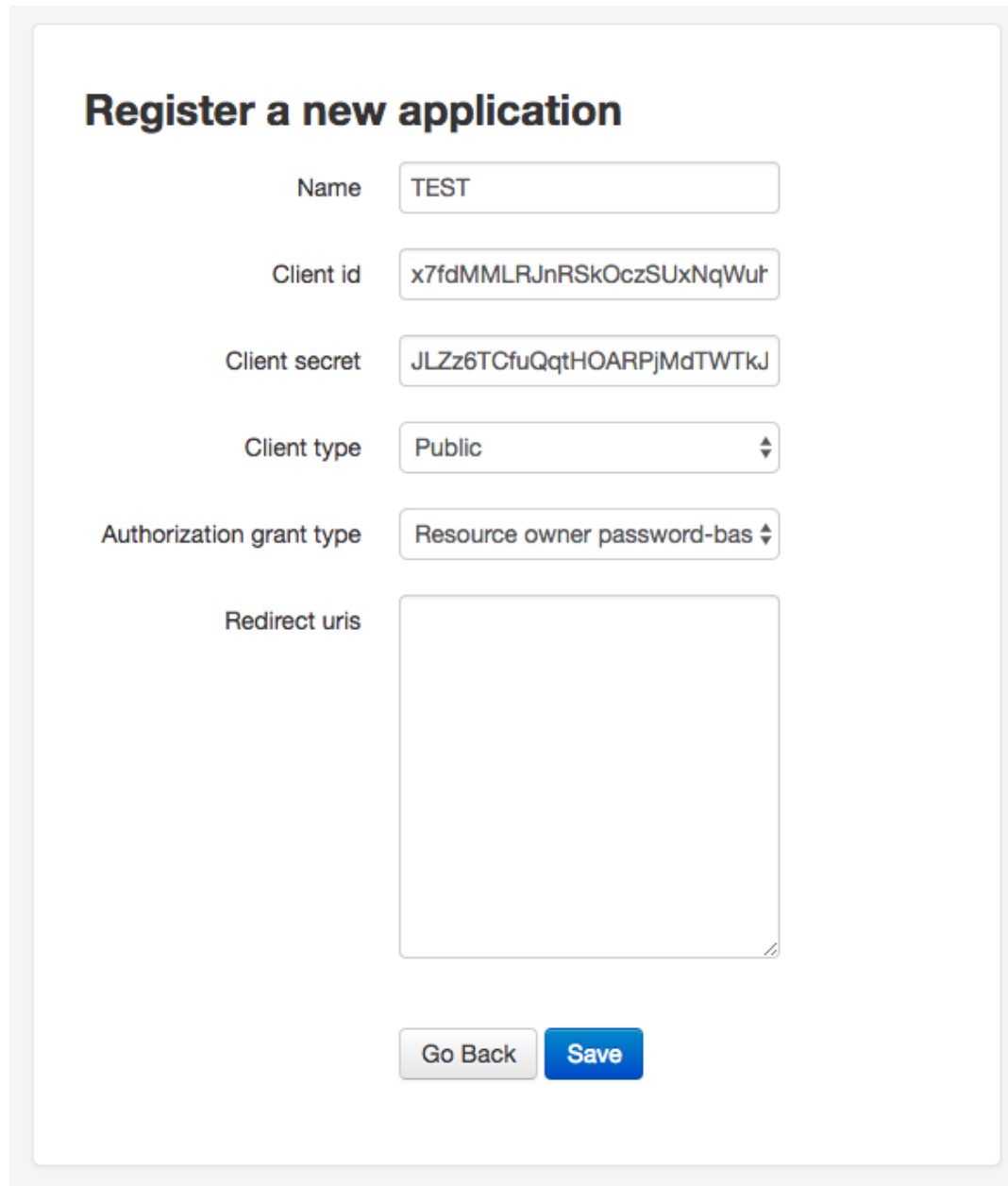
Register an OAuth Application

To allow others to connect to your Arches instance, you must create an OAuth client id and add it to your settings.

1. In a browser go to

```
http://<yourdomain:port>/o/applications/
```

2. Create a new application
3. Fill out the form with a **Name** of your choosing, and set **Client type** and **Authorization grant type** as shown in the image below.



Register a new application

Name

Client id

Client secret

Client type

Authorization grant type

Redirect uris

4. Copy the **Client id** and submit the form (you can access this id at any time).
5. In your Arches project's `settings.py` or `settings_local.py` file, set or add this variable

```
MOBILE_OAUTH_CLIENT_ID = "<your new Client id>"
```

Important:

- Only make one application, though you are technically allowed to make more.
 - An application is “owned” by whichever user created it, and will not be visible to other users.
-

Authentication

Most Arches API endpoints require an OAuth access token.

OAuth 2.0 is a simple and secure authentication mechanism. It allows applications to acquire an access token for Arches via a quick redirect to the Arches site. Once an application has an access token, it can access a user's resources on Arches. to authenticate with OAuth you must first [Register an OAuth Application](#).

POST /o/token

gets an OAuth token given a username, password, and client id

Note: You should only make this call once and store the returned token securely. You should not make this call per request or at any other high-frequency interval.

This token is to be used with clients registered with the “Resource Owner Password Credentials Grant” type see [Register an OAuth Application](#) for more information on registering an application

For additional information see <https://tools.ietf.org/html/rfc6749#section-4.3>

Form Parameters

- **username** – a users username (or email)
- **password** – a users password
- **grant_type** – “password”
- **client_id** – the registered applications client id, see [Register an OAuth Application](#)

Status Codes

- **401 Unauthorized** – there's no user or the user has been deactivated, or the client id is invalid

Example request:

```
curl -X POST http://localhost:8000/o/token/ -d "username=admin&password=admin&grant_type=password&client_id=onFiQSbPfgZpsUcl2fBvaaEHA58MKHavl3iuSaRf"
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "access_token": "TS3pE2bEXRCAkRls4IGKCVVa0Zv6FE",
  "token_type": "Bearer",
  "expires_in": 36000,
  "refresh_token": "y3rzXKf8dXdb25ayMMVIliligTkqEKr0",
  "scope": "read write"
}
```

returned when an invalid username or password is supplied

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{"error_description": "Invalid credentials given.", "error": "invalid_grant"}
```

returned when an invalid client id is supplied, or the registered client is not “public” or the grant type used to register the client isn’t “Resource Owner Password Credentials Grant”

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json

{"error": "invalid_client"}
```

Concepts

GET /rdm/concepts/{uuid:concept instance id}

gets a single rdm concept instance

Query Parameters

- **format** – {“json”}
- **indent** – number of spaces to indent json output
- **includesubconcepts** – option to include sub concepts in the return
- **includeparentconcepts** – option to include parent concepts in the return
- **includerelatedconcepts** – option to include related concepts in the return
- **depthlimit** – limit the number of subconcept layers to return if includesubconcepts is true
- **lang** – show subconcept results with specified language first

Request Headers

- **Authorization** – oAuth token for user authentication, see [/o/token](#)

Example request:

```
curl -H "Authorization: Bearer {token}" -X GET http://localhost:8000/rdm/concepts/
↪ {concept instance id}

curl -H "Authorization: Bearer zo41Q1IMgAW30xOroiCUxjv3yci80s" -X GET http://
↪ localhost:8000/rdm/concepts/5e04c83e-1ae3-42e8-ae31-4f7c25f737a5?format=json&
↪ indent=4
```

Example json response:

```
HTTP/1.0 200 OK
Content-Type: application/json

{
  "hassubconcepts": true,
  "id": "5e04c83e-1ae3-42e8-ae31-4f7c25f737a5",
  "legacyoid": "http://www.archesproject.org/5e04c83e-1ae3-42e8-ae31-4f7c25f737a5
↪ ",
  "nodetype": "Concept",
```

(continues on next page)

(continued from previous page)

```

"parentconcepts": [{
  "hassubconcepts": true,
  "id": "7b8e4771-2680-4004-9743-40ea78e8c2a9",
  "legacyoid": "http://www.archesproject.org/7b8e4771-2680-4004-9743-
↪40ea78e8c2a9",
  "nodetype": "ConceptScheme",
  "parentconcepts": [],
  "relatedconcepts": [],
  "relationshiptype": "hasTopConcept",
  "subconcepts": [],
  "values": [{
    "category": "label",
    "conceptid": "7b8e4771-2680-4004-9743-40ea78e8c2a9",
    "id": "b18048a9-4814-43f0-bb88-99fa22a42fbe",
    "language": "en-US",
    "type": "prefLabel",
    "value": "DISCO"
  }, {
    "category": "note",
    "conceptid": "7b8e4771-2680-4004-9743-40ea78e8c2a9",
    "id": "16ea8772-d5dd-481d-91a7-c09703718138",
    "language": "en-US",
    "type": "scopeNote",
    "value": "Concept scheme for managing Data Integration for Conservation
↪Science thesauri"
  }, {
    "category": "identifiers",
    "conceptid": "7b8e4771-2680-4004-9743-40ea78e8c2a9",
    "id": "9eaa8a10-e9f2-4ce3-ac8b-c4904097b4c9",
    "language": "en-US",
    "type": "identifier",
    "value": "http://www.archesproject.org/7b8e4771-2680-4004-9743-
↪40ea78e8c2a9"
  }]
}],
"relatedconcepts": [],
"relationshiptype": "",
"subconcepts": [{
  "hassubconcepts": false,
  "id": "0788acb1-9968-43e8-80f7-37b37e155f95",
  "legacyoid": "http://www.archesproject.org/0788acb1-9968-43e8-80f7-
↪37b37e155f95",
  "nodetype": "Concept",
  "parentconcepts": [{
    "hassubconcepts": false,
    "id": "5e04c83e-1ae3-42e8-ae31-4f7c25f737a5",
    "legacyoid": "http://www.archesproject.org/5e04c83e-1ae3-42e8-ae31-
↪4f7c25f737a5",
    "nodetype": "Concept",
    "parentconcepts": [],
    "relatedconcepts": [],
    "relationshiptype": "narrower",

```

(continues on next page)

(continued from previous page)

```

        "subconcepts": [],
        "values": []
    }],
    "relatedconcepts": [],
    "relationshipstype": "narrower",
    "subconcepts": [],
    "values": [{
        "category": "label",
        "conceptid": "0788acb1-9968-43e8-80f7-37b37e155f95",
        "id": "dd5c6d39-7bc4-438e-abe2-544b8ae06864",
        "language": "en-US",
        "type": "prefLabel",
        "value": "Artist"
    }, {
        "category": "identifiers",
        "conceptid": "0788acb1-9968-43e8-80f7-37b37e155f95",
        "id": "5f355975-29a7-4a53-8260-4093d63c1967",
        "language": "en-US",
        "type": "identifier",
        "value": "http://www.archesproject.org/0788acb1-9968-43e8-80f7-
↪37b37e155f95"
    }]
    },
    "values": [{
        "category": "label",
        "conceptid": "5e04c83e-1ae3-42e8-ae31-4f7c25f737a5",
        "id": "b75ca80a-3128-421d-ae2b-aacb7d12bbc7",
        "language": "en-US",
        "type": "prefLabel",
        "value": "DISCO Actor Types"
    }, {
        "category": "identifiers",
        "conceptid": "5e04c83e-1ae3-42e8-ae31-4f7c25f737a5",
        "id": "79d2e5d2-91fc-435d-869a-042c994d3481",
        "language": "en-US",
        "type": "identifier",
        "value": "http://www.archesproject.org/5e04c83e-1ae3-42e8-ae31-4f7c25f737a5"
    }]
}

```

Resources

GET /resources/

gets a paged list of resource instance ids in json-ld format

Query Parameters

- **page** – number specifying the page of results to return

Example request:

```
curl -X GET http://localhost:8000/resources/

curl -X GET http://localhost:8000/resources/?page=2
```

Example response:

```
HTTP/1.0 200 OK
Content-Type: application/json

{
  "@context": "https://www.w3.org/ns/ldp/",
  "@id": "",
  "@type": "ldp:BasicContainer",
  "ldp:contains": [
    "http://localhost:8000/resources/000000000-0000-0000-0000-0000000000100",
    "http://localhost:8000/resources/000000000-0000-0000-0000-0000000000101",
    "http://localhost:8000/resources/000ee2fe-4568-457b-960c-3e1ec3f53e10",
    "http://localhost:8000/resources/000fa53f-0f06-4648-a960-c42b8accd235",
    "http://localhost:8000/resources/00131129-7451-435d-aab9-33eb9031e6d1",
    "http://localhost:8000/resources/001b6c4b-f906-4df2-9fcd-b9fda95eed95",
    "http://localhost:8000/resources/0032990e-f8d6-4a7b-8032-d90d3c764b40",
    "http://localhost:8000/resources/003619ca-5fa7-4e75-b3b7-a62f40fe9419",
    "http://localhost:8000/resources/00366caa-3c00-4909-851d-0d650e62f820",
    "http://localhost:8000/resources/003874d7-8e73-4323-bddf-b893651e22c1",
    "http://localhost:8000/resources/003e56a0-d0eb-485f-b975-61faf2f22755",
    "http://localhost:8000/resources/0043a0be-c7be-4a35-9f6c-0ba80269caf4",
    "http://localhost:8000/resources/0060f35d-47a7-4f22-aaf3-fa2d0bd493f7",
    "http://localhost:8000/resources/0069dad8-41b6-4cad-8e54-f72fe8093550",
    "http://localhost:8000/resources/0069db14-a0c1-470e-abf7-eda7b56bf012"
  ]
}
```

GET /resources/{uuid:resource instance id}

gets a single resource instance

Query Parameters

- **format** – {“xml”, “json”, “json-ld”}
- **indent** – number of spaces to indent json output

Request Headers

- **Authorization** – OAuth token for user authentication, see [/o/token](#)
- **Accept** – optional alternative to “format”, {“application/xml”, “application/json”, “application/ld+json”}

Example request:

```
curl -H "Authorization: Bearer {token}" -X GET http://localhost:8000/resources/
↪{resource instance id}

curl -H "Authorization: Bearer zo41Q1IMgAW30x0roiCUxjv3yci80s" -X GET http://
↪localhost:8000/resources/00131129-7451-435d-aab9-33eb9031e6d1?format=json&indent=4
```

Example json response:

```

HTTP/1.0 200 OK
Content-Type: application/json

{
  "business_data": {
    "resources": [
      {
        "tiles": [
          {
            "data": {
              "e4b37f8a-343a-11e8-ab89-dca90488358a": "203 Boultham_
↪Park Road"
              "e4b4b7f5-343a-11e8-a681-dca90488358a": null,
            },
            "provisionaledits": null,
            "parenttile_id": null,
            "nodegroup_id": "e4b37f8a-343a-11e8-ab89-dca90488358a",
            "sortorder": 0,
            "resourceinstance_id": "99131129-7451-435d-aab9-33eb9031e6d1
↪",
            "tileid": "b72225a9-4e3d-47ee-8d94-52316469bc3f"
          },
          {
            "data": {
              "e4b3f15c-343a-11e8-a26b-dca90488358a": null,
              "e4b4ca3d-343a-11e8-ab73-dca90488358a": {
                "type": "FeatureCollection",
                "features": [
                  {
                    "geometry": {
                      "type": "Point",
                      "coordinates": [
                        -0.559288403624841,
                        53.2132233001817
                      ]
                    },
                    "type": "Feature",
                    "id": "c036e50a-4959-4b6f-93d0-2c03068c0948
↪",
                    "properties": {}
                  }
                ]
              },
              "provisionaledits": null,
              "parenttile_id": "4e40e6f3-8252-4439-831d-c371655cc4eb",
              "nodegroup_id": "e4b3f15c-343a-11e8-a26b-dca90488358a",
              "sortorder": 0,
              "resourceinstance_id": "99131129-7451-435d-aab9-33eb9031e6d1
↪",
              "tileid": "65199340-32c3-4936-a09e-7c5143552d15"
            },
          {

```

(continues on next page)

(continued from previous page)

```

        "data": {
            "e4b386eb-343a-11e8-82ef-dca90488358a": "Detached house,
↳ built by A B Sindell"
        },
        "provisionaledits": null,
        "parenttile_id": "8870d2d6-e179-4321-a8bb-543fd2db63c6",
        "nodegroup_id": "e4b386eb-343a-11e8-82ef-dca90488358a",
        "sortorder": 0,
        "resourceinstance_id": "99131129-7451-435d-aab9-33eb9031e6d1
↳ ",
        "tileid": "04bb7bef-1e6e-4228-bd87-3f0a129514a8"
    }
},
"resourceinstance": {
    "graph_id": "e4b3562b-343a-11e8-b509-dca90488358a",
    "resourceinstanceid": "99131129-7451-435d-aab9-33eb9031e6d1",
    "legacyid": "99131129-7451-435d-aab9-33eb9031e6d1"
}
}
]
}
}

```

PUT /resources/{uuid: graph id}/{uuid:resource instance id}

Note: Instead of identifying a graph by a UUID, one can also identify a graph by by a slug identifier. To get or set the slug for the graph, navigate to the root node of the [Graph Designer](#). A request using a slug identifier for a graph looks like: PUT /resources/{string: graph slug}/{uuid:resource instance id}

Updates a single resource instance

Query Parameters

- **format** – {"json-ld", "arches-json"}
- **indent** – number of spaces to indent json output

Request Headers

- **Authorization** – OAuth token for user authentication, see [/o/token](#)
- **Accept** – optional alternative to "format", {"application/json", "application/ld+json"}

Example request:

```

curl -H "Authorization: Bearer {token}" -X PUT -d {data in json-ld format} http://
↳ localhost:8000/resources/{graph id}/{resource instance id}

curl -H "Authorization: Bearer zo41Q1IMgAW30x0roiCUxjv3yci80s" -X PUT \
-d '{
    "@id": "http://localhost:8000/resource/47a1830c-74ec-11e8-bff6-14109fd34195",
    "@type": [
        "http://www.cidoc-crm.org/cidoc-crm/E18_Physical_Thing",
        "http://localhost:8000/graph/ab74af76-fa0e-11e6-9e3e-026d961c88e6"
    ]
}'

```

(continues on next page)

(continued from previous page)

```

    ],
    "http://www.cidoc-crm.org/cidoc-crm/P140i_was_attributed_by": {
      "@id": "http://localhost:8000/tile/1f7b4c8f-9932-47e4-9ec5-0284c77d893c/
↪node/677f236e-09cc-11e7-8ff7-6c4008b05c4c",
      "@type": "http://www.cidoc-crm.org/cidoc-crm/E15_Identifier_Assignment",
      "http://www.cidoc-crm.org/cidoc-crm/P1_is_identified_by": [
        {
          "@id": "http://localhost:8000/tile/6efb8ac0-623c-47cb-9846-
↪4a489c153683/node/677f303d-09cc-11e7-9aa6-6c4008b05c4c",
          "@type": "http://www.cidoc-crm.org/cidoc-crm/E41_Appellation",
          "http://www.cidoc-crm.org/cidoc-crm/P2_has_type": {
            "@id": "http://localhost:8000/tile/6efb8ac0-623c-47cb-9846-
↪4a489c153683/node/677f39a8-09cc-11e7-834a-6c4008b05c4c",
            "@type": "http://www.cidoc-crm.org/cidoc-crm/E55_Type",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "ech20ae9-
↪a457-4011-83bf-1c936e2d6b6a"
          },
          "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "Claudio"
        },
        {
          "@id": "http://localhost:8000/tile/b53f2aaa-348b-4b73-9ff9-
↪195090038c8b/node/677f303d-09cc-11e7-9aa6-6c4008b05c4c",
          "@type": "http://www.cidoc-crm.org/cidoc-crm/E41_Appellation",
          "http://www.cidoc-crm.org/cidoc-crm/P2_has_type": {
            "@id": "http://localhost:8000/tile/b53f2aaa-348b-4b73-9ff9-
↪195090038c8b/node/677f39a8-09cc-11e7-834a-6c4008b05c4c",
            "@type": "http://www.cidoc-crm.org/cidoc-crm/E55_Type",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "81dd62d2-
↪6701-4195-b74b-8057456bba4b"
          },
          "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "Alejandro"
        }
      ],
      "http://www.cidoc-crm.org/cidoc-crm/P2_has_type": {
        "@id": "http://localhost:8000/tile/e818ecc5-8bde-4978-baca-2206a5bbf509/
↪node/677f2c0f-09cc-11e7-b412-6c4008b05c4c",
        "@type": "http://www.cidoc-crm.org/cidoc-crm/E55_Type",
        "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "e4699732-efee-46c0-
↪87e1-3f0a930a43db"
      }
    }
  }' \
'http://localhost:8000/resources/00131129-7451-435d-aab9-33eb9031e6d1?format=json-
↪ld&indent=4'

```

Example json response:

```

HTTP/1.0 200 OK
Content-Type: application/json

{
  "@id": "http://localhost:8000/resource/47a1830c-74ec-11e8-bff6-14109fd34195",

```

(continues on next page)

(continued from previous page)

```

"@type": [
  "http://www.cidoc-crm.org/cidoc-crm/E18_Physical_Thing",
  "http://localhost:8000/graph/ab74af76-fa0e-11e6-9e3e-026d961c88e6"
],
"http://www.cidoc-crm.org/cidoc-crm/P140i_was_attributed_by": {
  "@id": "http://localhost:8000/tile/1f7b4c8f-9932-47e4-9ec5-0284c77d893c/
↪node/677f236e-09cc-11e7-8ff7-6c4008b05c4c",
  "@type": "http://www.cidoc-crm.org/cidoc-crm/E15_Identifier_Assignment",
  "http://www.cidoc-crm.org/cidoc-crm/P1_is_identified_by": [
    {
      "@id": "http://localhost:8000/tile/6efb8ac0-623c-47cb-9846-
↪4a489c153683/node/677f303d-09cc-11e7-9aa6-6c4008b05c4c",
      "@type": "http://www.cidoc-crm.org/cidoc-crm/E41_Appellation",
      "http://www.cidoc-crm.org/cidoc-crm/P2_has_type": {
        "@id": "http://localhost:8000/tile/6efb8ac0-623c-47cb-9846-
↪4a489c153683/node/677f39a8-09cc-11e7-834a-6c4008b05c4c",
        "@type": "http://www.cidoc-crm.org/cidoc-crm/E55_Type",
        "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "ech20ae9-
↪a457-4011-83bf-1c936e2d6b6a"
      },
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "Claudio"
    },
    {
      "@id": "http://localhost:8000/tile/b53f2aaa-348b-4b73-9ff9-
↪195090038c8b/node/677f303d-09cc-11e7-9aa6-6c4008b05c4c",
      "@type": "http://www.cidoc-crm.org/cidoc-crm/E41_Appellation",
      "http://www.cidoc-crm.org/cidoc-crm/P2_has_type": {
        "@id": "http://localhost:8000/tile/b53f2aaa-348b-4b73-9ff9-
↪195090038c8b/node/677f39a8-09cc-11e7-834a-6c4008b05c4c",
        "@type": "http://www.cidoc-crm.org/cidoc-crm/E55_Type",
        "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "81dd62d2-
↪6701-4195-b74b-8057456bba4b"
      },
      "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "Alejandro"
    }
  ],
  "http://www.cidoc-crm.org/cidoc-crm/P2_has_type": {
    "@id": "http://localhost:8000/tile/e818ecc5-8bde-4978-baca-2206a5bbf509/
↪node/677f2c0f-09cc-11e7-b412-6c4008b05c4c",
    "@type": "http://www.cidoc-crm.org/cidoc-crm/E55_Type",
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#value": "e4699732-efee-46c0-
↪87e1-3f0a930a43db"
  }
}
}

```

DELETE /resources/{uuid:resource instance id}

deletes a single resource instance

Request Headers

- **Authorization** – OAuth token for user authentication, see [/o/token](#)

Example request:

```
curl -H "Authorization: Bearer {token}" -X DELETE http://localhost:8000/resources/
↳{resource instance id}

curl -H "Authorization: Bearer zo41Q1IMgAW30x0roiCUxjv3yci80s" -X DELETE http://
↳localhost:8000/resources/00131129-7451-435d-aab9-33eb9031e6d1
```

Example response:

```
HTTP/1.0 200 OK
```

Activity Stream

GET /history/

gets a JSON-LD representation of the collection that comprises the changes made (Create, Update, Delete) to Arches resources.

Request Headers

- [Authorization](#) – OAuth token for user authentication, see [/o/token](#)

Example request:

```
curl -X GET http://localhost:8000/history/
```

Example response:

```
HTTP/1.0 200 OK
Content-Type: application/json

{
  "@context": "https://www.w3.org/ns/activitystreams",
  "type": "OrderedCollection",
  "id": "http://localhost:8000/history/",
  "totalItems": 7,
  "first": {
    "type": "OrderedCollectionPage",
    "id": "http://localhost:8000/history/1"
  },
  "last": {
    "type": "OrderedCollectionPage",
    "id": "http://localhost:8000/history/1"
  }
}
```

GET /history/{int: page number}

gets a single 'OrderedCollectionPage' JSON-LD representation for a given page number

Request Headers

- [Authorization](#) – OAuth token for user authentication, see [/o/token](#)

Example request:


```
curl -H "Authorization: Bearer {token}" -X GET http://localhost:8000/history/{page_
↪number}

curl -H "Authorization: Bearer zo41Q1IMgAW30x0roiCUxjv3yci80s" -X GET http://
↪localhost:8000/history/1
```

Example json response:

```
HTTP/1.0 200 OK
Content-Type: application/json

{
  "@context": "https://www.w3.org/ns/activitystreams",
  "type": "OrderedCollectionPage",
  "id": "http://localhost:8000/history/1",
  "partOf": {
    "totalItems": 7,
    "type": "OrderedCollection",
    "id": "http://localhost:8000/history/"
  },
  "orderedItems": [
    {
      "endTime": "2019-06-20T17:38:56Z",
      "type": "Create",
      "actor": {
        "url": "http://localhost:8000/user/1",
        "tag": null,
        "type": "Person",
        "name": "", ""
      },
      "object": {
        "url": "http://localhost:8000/resources/47b179f0-9382-11e9-b0f5-
↪0242ac120003",
        "type": "http://www.cidoc-crm.org/cidoc-crm/E33_Linguistic_Object"
      }
    },
    {
      "endTime": "2019-06-20T17:38:57Z",
      "type": "Update",
      "actor": {
        "url": "http://localhost:8000/user/1",
        "tag": "admin",
        "type": "Person",
        "name": "", ""
      },
      "object": {
        "url": "http://localhost:8000/resources/47b179f0-9382-11e9-b0f5-
↪0242ac120003",
        "type": "http://www.cidoc-crm.org/cidoc-crm/E33_Linguistic_Object"
      }
    },
    {
      "endTime": "2019-06-20T17:39:04Z",
```

(continues on next page)

(continued from previous page)

```

        "type": "Update",
        "actor": {
            "url": "http://localhost:8000/user/1",
            "tag": "admin",
            "type": "Person",
            "name": "", ""
        },
        "object": {
            "url": "http://localhost:8000/resources/47b179f0-9382-11e9-b0f5-
↪0242ac120003",
            "type": "http://www.cidoc-crm.org/cidoc-crm/E33_Linguistic_Object"
        }
    },
    {
        "endTime": "2019-06-20T17:39:13Z",
        "type": "Create",
        "actor": {
            "url": "http://localhost:8000/user/1",
            "tag": null,
            "type": "Person",
            "name": "", ""
        },
        "object": {
            "url": "http://localhost:8000/resources/514796f2-9382-11e9-9e60-
↪0242ac120003",
            "type": "http://www.cidoc-crm.org/cidoc-crm/E22_Man-Made_Object"
        }
    },
    {
        "endTime": "2019-06-20T17:39:13Z",
        "type": "Update",
        "actor": {
            "url": "http://localhost:8000/user/1",
            "tag": "admin",
            "type": "Person",
            "name": "", ""
        },
        "object": {
            "url": "http://localhost:8000/resources/514796f2-9382-11e9-9e60-
↪0242ac120003",
            "type": "http://www.cidoc-crm.org/cidoc-crm/E22_Man-Made_Object"
        }
    },
    {
        "endTime": "2019-06-20T17:39:15Z",
        "type": "Update",
        "actor": {
            "url": "http://localhost:8000/user/1",
            "tag": "admin",
            "type": "Person",
            "name": "", ""
        },
    },

```

(continues on next page)

(continued from previous page)

```

        "object": {
          "url": "http://localhost:8000/resources/47b179f0-9382-11e9-b0f5-
↪0242ac120003",
          "type": "http://www.cidoc-crm.org/cidoc-crm/E33_Linguistic_Object"
        },
        {
          "endTime": "2019-06-20T17:39:24Z",
          "type": "Update",
          "actor": {
            "url": "http://localhost:8000/user/1",
            "tag": "admin",
            "type": "Person",
            "name": "", ""
          },
          "object": {
            "url": "http://localhost:8000/resources/47b179f0-9382-11e9-b0f5-
↪0242ac120003",
            "type": "http://www.cidoc-crm.org/cidoc-crm/E33_Linguistic_Object"
          }
        }
      ]
    }
  }
}

```

Mobile Projects

GET /mobileprojects

get a list of mobile data collection projects that a user has been invited to participate in

Example request:

```

curl -H "Authorization: Bearer {token}" -X GET http://localhost:8000/mobileprojects

curl -H "Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ.-xN_
↪h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM" -X GET http://localhost:8000/
↪mobileprojects

```

Example response:

```

HTTP/1.0 200 OK
Content-Type: application/json

[
  {
    "active": true,
    "bounds": "MULTIPOLYGON EMPTY",
    "cards": [],
    "createdby_id": 1,
    "datadownloadconfig": {
      "count": 1000,

```

(continues on next page)

(continued from previous page)

```

        "custom": null,
        "download": false,
        "resources": []
    },
    "description": "A description of this project.",
    "enddate": "2018-03-16",
    "groups": [
        6
    ],
    "id": "e3d95999-2323-11e8-894b-14109fd34195",
    "lasteditedby_id": 1,
    "name": "Forbidden Project",
    "startdate": "2018-03-04",
    "tilecache": "",
    "users": [
        1
    ]
}
]

```

Request Headers

- **Authorization** – JWT (JSON web token) for user authentication, see /auth/get_token

GeoJSON

GET /geojson

returns a GeoJSON representation of resource instance data; this will include metadata properties when using paging for “_page” (number) and “_lastPage” (boolean). Returned features will include integer ids that are only assured to be unique per request.

NOTE: when not using the “use_uuid_names” parameter, field names will use the export field name provided for a given node (via the Graph Designer). If the export field name is not defined, the API will attempt to create a suitable field name from the node name. Property names that clash as a result of the above, or shortening via “field_name_length” will have their values joined together.

WARNING: including primary names has a big impact on performance and is best deferred to an additional request

Query Parameters

- **resourceid** – optional comma delimited list of resource instance UUIDs to filter feature data on
- **nodeid** – optional node UUID to filter feature data on
- **tileid** – optional tile UUID to filter feature data on
- **nodegroups** – optional comma delimited list of nodegroup UUIDs from which to include tile data as properties.
- **precision** – optional number of decimal places returned in coordinate values; used to constrain resultant data volume
- **field_name_length** – optional number to limit property field length to

- **use_uuid_names** – include this parameter to return tile property names as node UUIDs.
- **include_primary_name** – include this parameter to include resource instance primary names in feature properties.
- **use_display_values** – include this parameter to return tile values processed to be human readable
- **include_geojson_link** – include this parameter to include a link to this specific feature in its properties fit for reuse later
- **indent** – optional number of spaces with which to indent the JSON return (ie “pretty print”)
- **type** – optional geometry type name to filter features on
- **limit** – optional number of tiles to process; used to page data. NOTE: as paging is per tile, the count of features in the response may differ from this limit value
- **page** – optional number of page (starting with 1) to return; used in conjunction with “limit”

Example request:

```
curl -X GET http://localhost:8000/geojson?nodegroups=8d41e4ab-a250-11e9-87d1-
→00224800b26d,8d41e4c0-a250-11e9-a7e3-00224800b26d&nodeid=8d41e4d6-a250-11e9-accd-
→00224800b26d&use_display_values=true&indent=2&limit=3
```

Example response:

```
HTTP/1.0 200 OK
Content-Type: application/json
```

```
{
  "_lastPage": false,
  "_page": 1,
  "features": [{
    "geometry": {
      "coordinates": [
        -0.09160837,
        51.529378348
      ],
      "type": "Point"
    },
    "id": 1,
    "properties": {
      "application_type": "Enquiry",
      "consultation_status": "Dormant",
      "consultation_type": "Post-Application",
      "development_type": "Mixed Use",
      "name": "Consultation for 93 Mendota Alley",
      "resourceinstanceid": "aa7ecf38-ab81-4e08-bb74-cfdd1e339ea2",
      "tileid": "4e4d8fe8-3ee9-4ddc-9613-fffc1511bd58"
    },
    "type": "Feature"
  }, {
    "geometry": {
      "coordinates": [
        -0.090902277,
        51.533642427
```

(continues on next page)

(continued from previous page)

```

        ],
        "type": "Point"
    },
    "id": 2,
    "properties": {
        "application_type": "Listed Building Consent",
        "consultation_status": "Completed",
        "consultation_type": "Condition Application",
        "development_type": "Land restoration",
        "name": "Consultation for 57359 Fieldstone Way",
        "resourceinstanceid": "2cf195f8-805b-4f97-9133-cbd94bf5a01f",
        "tileid": "6e3009d4-4022-4510-8e42-504b5bc20b74"
    },
    "type": "Feature"
}, {
    "geometry": {
        "coordinates": [
            -0.088202575,
            51.533347841
        ],
        "type": "Point"
    },
    "id": 3,
    "properties": {
        "application_type": "Listed Building Consent",
        "consultation_status": "Aborted",
        "consultation_type": "Post-Application",
        "development_type": "Road construction",
        "name": "Consultation for 3660 Kim Court",
        "resourceinstanceid": "eefa863a-53e4-404a-89b4-6213b46b2b55",
        "tileid": "99395221-dd7f-4a06-8d87-5f5703501ab5"
    },
    "type": "Feature"
}],
"type": "FeatureCollection"
}

```

Command Line Reference

- *Installation Commands*
- *ElasticSearch Management*
- *Import Commands*
- *Export Commands*
- *Managing Functions, DataTypes, Widgets, and Card Components*
- *Other Useful Django Commands*

This page serves as a quick reference guide for working with Arches through a command prompt. Along with default Django commands, a good deal of Arches operations have been added to `manage.py`. In a command prompt, [activate your virtual environment](Dev-Installation#4-activate-the-virtual-environment), then run the following commands

from your root app directory (the one that contains `manage.py`).

All file or directory path parameters (`-s`, `-c`, `-d`) should be absolute paths.

Installation Commands

installing from a local repo clone

```
pip install -e .
```

-e This argument with the value `.` indicates to pip that it should link the local directory with the virtual environment.

Installs Arches into your virtual environment from a local clone of the [archesproject/arches](#) repo, or your own fork of that repo. To do this properly, create a new virtual environment and activate it, clone the repo you want, enter that repo's root directory, and then run the command. Also, this command must be followed by:

```
pip install -r arches/install/requirements.txt
```

in order to properly install all of Arches' python requirements. Make sure to use `\` instead of `/` on Windows.

creating an Arches project

```
arches-project create <name_of_project> [{-d|--directory} <directory_name>]
```

-d, --directory (Optional) The name of the directory you'd like your new project located in.

creating (or recreating) the database

```
python manage.py setup_db
```

Deletes and recreates the database, as defined by `settings.DATABASES['default']`. Likewise, **this command will remove all existing data.**

loading a package into a project

```
python manage.py packages -o load_package -s source_of_package [-db]
```

-db Add this boolean argument to force the destruction and recreation of your database before loading the package.

The source (`-s`) of a package can be either a path to a local directory, the location of a local zipfile containing a package, or the url to a github repo archive that contains a package. For example, loading the sample package from where it resides in github would just be:

```
python manage.py packages -o load_package -s https://github.com/archesproject/arches-
↪example-pkg/archive/master.zip
```

ElasticSearch Management

reindex the database

Note that commands using `python manage.py es [command]` require ElasticSearch to be running.

```
python manage.py es reindex_database
```

This single command wraps the three following commands (each of which can be run individually if desired).

```
python manage.py es delete_indexes
python manage.py es setup_indexes
python manage.py es index_database
```

Important: If `DEBUG = True`, memory usage will continuously increase during indexing, because Django stores all db queries in memory, and a lot of them happen during indexing. Be wary of this during development when indexing large databases, or on servers with small memory provisions (you may want to temporarily set `DEBUG = False`).

Starting with version 7.4, you can add the `-rd` or `--recalculate-descriptors` flag to the reindex management command to force resource instance primary descriptors to be recalculated prior to reindexing. See below:

```
python manage.py es reindex_database --recalculate-descriptors
```

register a custom index

```
python manage.py es add_index --name {index name}
```

See [Adding a Custom Index](#)

Import Commands

Import Resource Models or Branches in archesjson format

```
python manage.py packages -o import_graphs [-s path_to_json_directory_or_file]
```

<code>-s</code>	Path to the source file you are importing. If not specified, the command will look to <code>settings.RESOURCE_GRAPH_LOCATIONS</code> for directory paths
-----------------	--

Import reference data in skos/rdf format

```
python manage.py packages -o import_reference_data -s 'path_to_rdf_file' [-ow {'overwrite'
↪'|'ignore'}}] [-st {'stage'|'keep'}]
```


Import business data

```
python manage.py packages -o import_business_data -s 'path_to_source_file' [-c 'path_to_
↪mapping_file'] [-ow '{overwrite}'|'append'}] [--create_concepts {'create'|'append'}] [--
↪bulk_load]
```

- c** The path to the mapping file. The mapping file tells Arches how to map the columns from your csv file to the nodes in your resource graph. This option is required if there is not a mapping file named the same as the business data file and in the same directory with extension ‘.mapping’ instead of ‘.csv’ or ‘.json’.
- ow** Determines how resources with duplicate ResourceIDs will be handled: **append** adds more tile data to an existing resource; **overwrite** replaces any existing resource with the imported data. This option only applies to CSV import. **JSON import always overwrites.**
- bulk, --bulk_load** Bulk load values into the database. By setting this flag the system will use Django’s **bulk_create** operation. The model’s **save()** method will not be called, and the **pre_save** and **post_save** signals will not be sent.
- create_concepts** Creates or appends concepts and collections to your rdm according to the option you select. **create** will create concepts and collections and associate them to the mapped nodes. **append** will append concepts to the existing collections assigned to the mapped nodes and create collections for nodes that do not have an assigned collection.

See also:

See *CSV Import* for CSV formatting requirements.

Import resource to resource relations

```
python manage.py packages -o import_business_data_relations -s 'path_to_relations_file'
```

See *Importing Resource Relations*

Export Commands

export branch or resource model schema

```
python manage.py packages -o export_graphs -d 'path_to_destination_directory' -g uuid/
↪branches/resource_models/all
```

- o** packages operation, in this case **export_graphs**
- d** Absolute path to destination directory
- g** UUID of specific graph, or **branches** for all branches, **resource_models** for all resource models, or **all** for everything.

Exports Resource Models and/or Branches. Note that sometimes (as in this case) Resource Models and Branches are generically called “graphs”.

export business data to csv or json

```
python manage.py packages -o export_business_data -d 'path_to_destination_directory' -f
↪ 'csv' or 'json' [-c 'path_to_mapping_file' -g 'resource_model_uuid' -single_file]
```

-o	<i>packages</i> operation, in this case <code>export_business_data</code>
-d	Absolute path to destination directory
-f	Export format, must be csv or json
-c	(required for csv) Absolute path to the mapping file you would like to use for your csv export.
-single_file	(optional for csv) Use this parameter if you'd like to export your grouped data to the same csv file as the rest of your data.
-g	(required for json, optional for csv) The resource model UUID whose instances you would like to export.

Exports business data to csv or json depending on the `-f` parameter specified. For csv export a mapping file is required. The exporter will export all resources of the type indicated in the `resource_model_id` property of the mapping file and the `-g` parameter will be ignored. For json export no mapping file is required, instead a resource model uuid should be passed into the `-g` command.

Note that in a Windows command prompt, you may need to replace `'` with `"`.

export business data to shapefile

```
python manage.py export shp -t 'name_of_db_view' -d 'output_directory'
```

-t	A resource instance database view
-d	The destination directory for point, line, and polygon shapefiles, created when the command is run.

business data export examples

```
python manage.py packages -o export_business_data -f 'csv' -c 'path_to_mapping_file'
```

Exports all business data of the resource model indicated in the mapping file. Two files are created. The first file contains one row per resource (if you resources all have the same geometry type this file can be used to create a shape file in QGIS or other program). The second file contains the grouped attributes of your resources (for instance, alternate names, additional classifications, etc.).

```
python manage.py packages -o export_business_data -f 'json' -g 'resource_model_id'
```

-f	'json' or 'csv'
-----------	-----------------

Exports all business data of the passed in `resource_model_id` to the specified file format. Take a look at the `RESOURCE_FORMATTERS` dictionary in Arches' `settings.py` for some other interesting options.

Other Data Management Commands

```
python manage.py resources remove_resources [-g graph_id] [-y]
```

- g** A Graph UUID to remove all the resource instances of.
- y** Forces this command to run without interactive confirmation.

Removes all resources from your database, but leaves the all resources models, branches, thesauri, and collections intact.

```
python manage.py packages -o create_mapping_file -d 'path_to_destination_directory' -g
↪ 'comma separated graph uuids'
```

- d** Path to directory to place the output in.
- g** One or more graph UUIDs to create a mapping for.

This mimics the ‘Create Mapping File’ command from the Arches Designer UI.

```
python manage.py packages -o import_mapping_file -s 'path_to_mapping_file'
```

Imports a mapping file for a particular resource model. This will be used as the export mapping file for a resource by default (e.g. for search export).

Ontology Commands

load an ontology

```
python manage.py load_ontology [-s <path to ontology directory>]
```

- s** Path to new ontology directory to load

Managing Functions, DataTypes, Widgets, and Card Components

To learn how to build new Functions, DataTypes, Card Components, or Widgets, please see *Functions*, *Widgets*, *Card Components*, or *Datatypes*. **Note that when importing Widgets and associated DataTypes, Widgets must be registered first.**

function commands

list registered functions

```
python manage.py fn list
```

Lists all currently registered functions.

registering functions

```
python manage.py fn register --source path/to/your/function.py
```

Register a newly created function. These .py files should sit in your projects functions directory.

unregistering functions

```
python manage.py fn unregister -n 'Sample Function'
```

Unregister a function. Use the function name that is returned by `fn list`.

datatype commands

list registered datatypes

```
python manage.py datatype list
```

Lists all currently registered datatypes.

registering and updating datatypes

```
python manage.py datatype register --source /Users/me/Documents/projects/mynewproject/  
↪mynewproject/datatypes/wkt_point.py
```

Registers a new datatype, in this example as defined in `wkt_point.py`.

```
python manage.py datatype update --source /Users/me/Documents/projects/mynewproject/  
↪mynewproject/datatypes/wkt_point.py
```

Updates a datatype, necessary anytime changes are made to your datatype's properties.

`-source` Location of the `.py` file that defines the datatype.

unregister a datatype

```
python manage.py datatype unregister -d 'wkt-point'
```

Unregisters a datatype, in this example a datatype named `wkt-point`.

-d Name of datatype to unregister. Use the datatype name that is returned by `datatype list`.

widget commands

All widget-related commands are identical to those for datatypes, just substitute `widget` for `datatype`. Also note that where datatypes are defined in `.py` files, widgets are defined in `.json` files.

card component commands

All component-related commands are identical to those for widgets, just substitute `card_component` for `widget`. JSON files are used to register Card Components.

Creating Map Layers

See *Creating New Map Layers* for file format requirements and other in-depth information.

MapBox

```
python manage.py packages -o add_mapbox_layer -j /path/to/mapbox_style.json -n "New↵
↵MapBox Layer" [{-b|--is_basemap}] [{-i|--layer_icon} 'icon_class']
```

-j The path to the Mapbox JSON file

-n The name of the Mapbox layer

Other Useful Django Commands

Run the django webserver

```
python manage.py runserver
```

Run the Django dev server. Add `0.0.0.0:8000` to explicitly set the host and port, which may be necessary when using remote servers, like an AWS EC2 instance. More about [runserver](#).

collect static files

```
python manage.py collectstatic
```

Collects all static files and places them in a single directory. Generally only necessary in production. Also allows all static files to be [hosted on another server](#)).

Django's full `manage.py` commands are documented [here](#).

Data Model

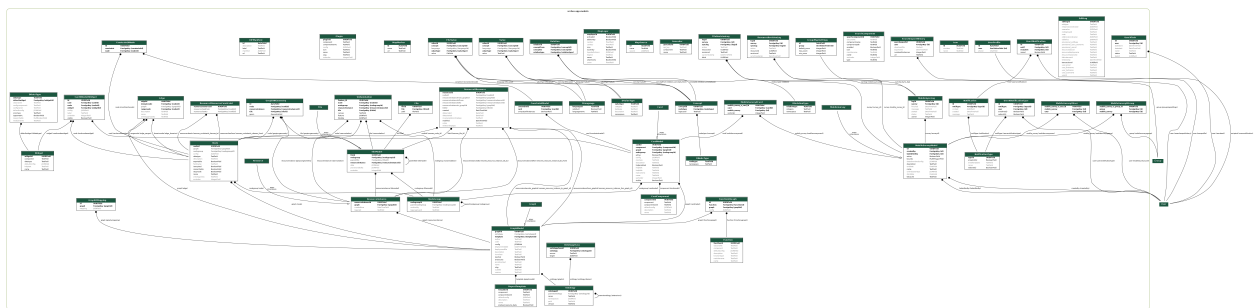


Fig. 32: Arches data model.

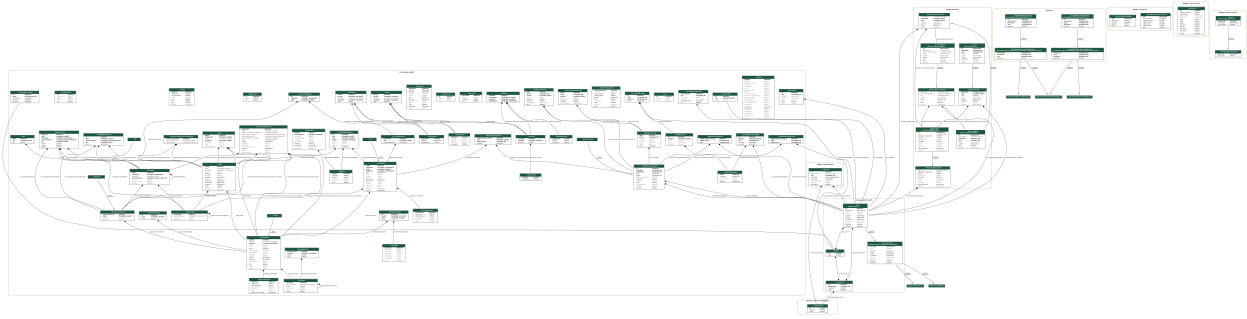


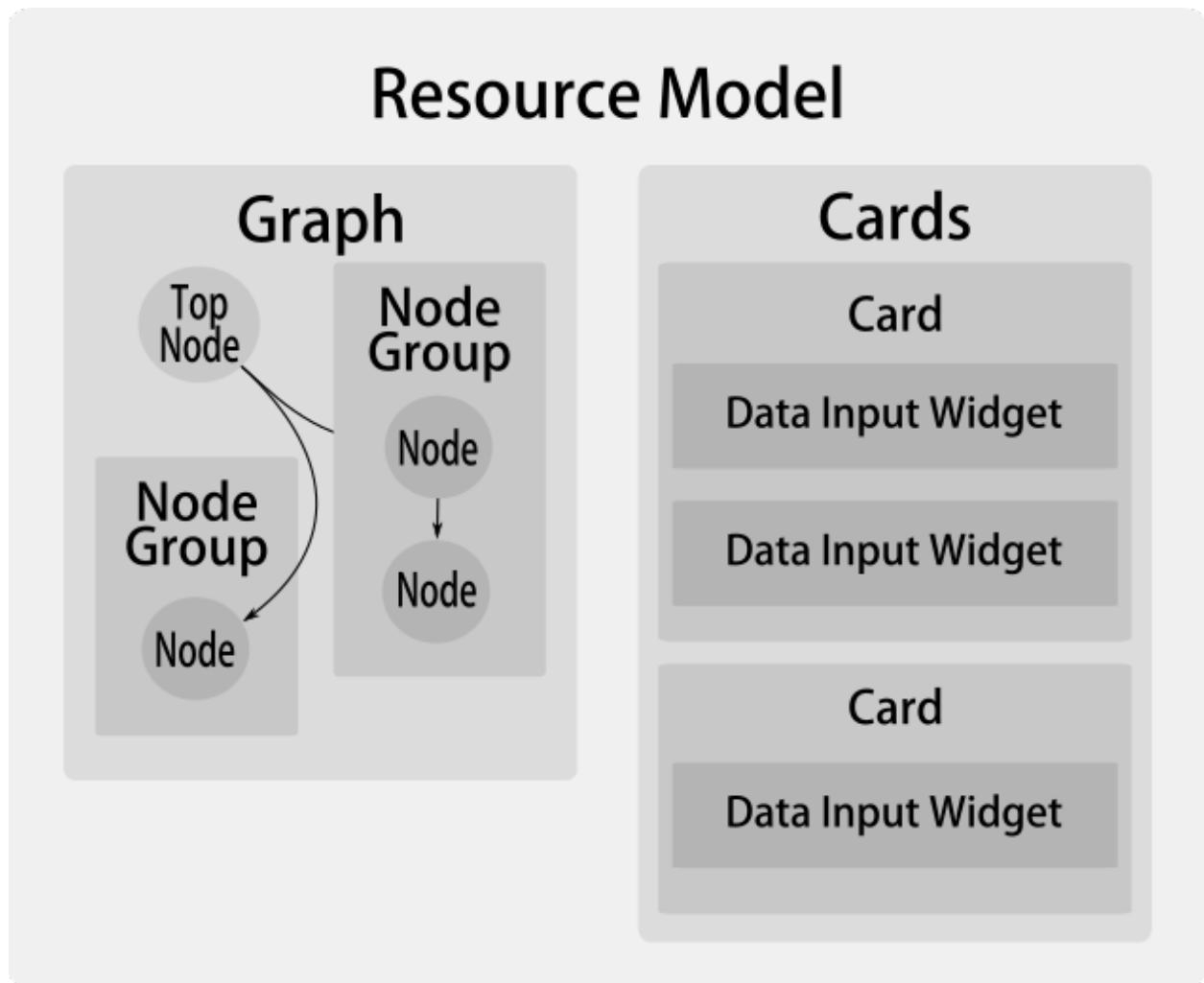
Fig. 33: Full model of all apps.

Resource Model Overview

Resources in an Arches database are separated into distinct Resource Models designed to represent a kind of physical real-world resource, such as a historic artifact or event. In the technical sense, the term **Resource Model** refers collectively to the following user-facing elements in Arches:

1. A Graph data structure representing a physical real-world resource, such as a building, a public figure, a website, an archaeological site, or a historic document.
2. A set of Cards to collect and display data associated with instances of this Resource Model.

The relationships among these components and their dependencies are visualized below:



The Arches logical model has been developed to support this modular construction, and the relevant models are described below as they pertain to the graph, UI components, and the resource data itself (not illustrated above).

Note: In the UI you will see a distinction between “Resource Models” and “Branches”, but underneath these are both made from instances of the *Graph model*. The primary difference between the two is the `isresource` property, which is set to `True` for a Resource Model.

Branches are used for records that might appear in multiple Resource Models, such as a person or place. Branches can be included as children of any Ontology-permitted Node in a Resource Model.

Controllers

Arches platform code defines base classes for some of its core data models, and uses [proxy models](#) to implement their controllers. In smaller classes, “controller” code is included with the data model class. This documentation primarily discusses the models, but controller behavior is discussed where relevant to how the models are used, and all models are referred to by their more succinct “controller” name.

Model	Controller
<i>ResourceInstance</i>	Resource
<i>CardModel</i>	Card
<i>TileModel</i>	Tile
<i>GraphModel</i>	Graph

Note: *ResourceInstance* breaks the implicit naming convention above because the term “Resource Model” refers to a specific Arches construct, as explained in the [Resource Model Overview](#) above.

Graph Definition

A Graph is a collection of [NodeGroups](#), [Nodes](#), and [Edges](#) which connect the Nodes.

Note: This definition does not include UI models and attributes, which are discussed [below](#).

In the Arches data model, Nodes represent their graph data structure namesakes, sometimes called *vertices*. A Node does the work of defining the Graph data structure in conjunction with one or more Edges, and sometimes collecting data.

NodeGroups are an Arches feature used to represent a group of one or more Nodes that collect data. NodeGroups can be nested, creating a metadata structure which is used to display the graph in the UI and collect related information together.

A NodeGroup exists for every Node that collects data, and both contains and shares its UUID with that node (see [naming conventions for references](#)). NodeGroups with more than one member Node are used to collect composite or semantically-related information. For example, a NodeGroup for a Node named `Name.E1` may contain a `Name.Type.E55` Node. This way, a Graph with this NodeGroup may store Names with multiple “types”, always collecting the information together.

NodeGroups are used to create [Cards](#), and this is done based on the `cardinality` property. Therefore, not every NodeGroup will be used to create a Card, which allows NodeGroups to exist within other NodeGroups. The `parentnodegroup` property is used to record this nesting.

A user-defined Function may be registered and then associated with a Graph in order to extend the behavior of Arches. For more information, see [here](#).

GraphModel

```

class GraphModel(models.Model):
    graphid = models.UUIDField(primary_key=True, default=uuid.uuid1)
    name = models.TextField(blank=True, null=True)
    description = models.TextField(blank=True, null=True)
    deploymentfile = models.TextField(blank=True, null=True)
    author = models.TextField(blank=True, null=True)
    deploymentdate = models.DateTimeField(blank=True, null=True)
    version = models.TextField(blank=True, null=True)
    isresource = models.BooleanField()
    isactive = models.BooleanField()
    iconclass = models.TextField(blank=True, null=True)
    color = models.TextField(blank=True, null=True)
    subtitle = models.TextField(blank=True, null=True)
    ontology = models.ForeignKey('Ontology', db_column='ontologyid', related_name='graphs
↪', null=True, blank=True)
    functions = models.ManyToManyField(to='Function', through='FunctionXGraph')
    jsonldcontext = models.TextField(blank=True, null=True)
    template = models.ForeignKey(
        'ReportTemplate',
        db_column='templateid',
        default='500000000-0000-0000-0000-0000000000001'
    )
    config = JSONField(db_column='config', default={})

    @property
    def disable_instance_creation(self):
        if not self.isresource:
            return _('Only resource models may be edited - branches are not editable')
        if not self.isactive:
            return _('Set resource model status to Active in Graph Designer')
        return False

    def is_editable(self):
        result = True
        if self.isresource:
            resource_instances = ResourceInstance.objects.filter(graph_id=self.graphid).
↪count()
            result = False if resource_instances > 0 else True
            if settings.OVERRIDE_RESOURCE_MODEL_LOCK == True:
                result = True
        return result

    class Meta:
        managed = True
        db_table = 'graphs'

```

Node

```

class Node(models.Model):
    """
    Name is unique across all resources because it ties a node to values within tiles.
    ↳ Recommend prepending resource class to node name.

    """

    nodeid = models.UUIDField(primary_key=True, default=uuid.uuid1)
    name = models.TextField()
    description = models.TextField(blank=True, null=True)
    istopnode = models.BooleanField()
    ontologyclass = models.TextField(blank=True, null=True)
    datatype = models.TextField()
    nodegroup = models.ForeignKey(NodeGroup, db_column='nodegroupid', blank=True,
    ↳ null=True)
    graph = models.ForeignKey(GraphModel, db_column='graphid', blank=True, null=True)
    config = JSONField(blank=True, null=True, db_column='config')
    issearchable = models.BooleanField(default=True)
    isrequired = models.BooleanField(default=False)
    sortorder = models.IntegerField(blank=True, null=True, default=0)

    def get_child_nodes_and_edges(self):
        """
        gather up the child nodes and edges of this node

        returns a tuple of nodes and edges

        """
        nodes = []
        edges = []
        for edge in Edge.objects.filter(domainnode=self):
            nodes.append(edge.rangenode)
            edges.append(edge)

            child_nodes, child_edges = edge.rangenode.get_child_nodes_and_edges()
            nodes.extend(child_nodes)
            edges.extend(child_edges)
        return (nodes, edges)

    @property
    def is_collector(self):
        return str(self.nodeid) == str(self.nodegroup_id) and self.nodegroup is not None

    def get_relatable_resources(self):
        relatable_resource_ids = [
            r2r.resourceclassfrom for r2r in Resource2ResourceConstraint.objects.
    ↳ filter(resourceclassto_id=self.nodeid)]
        relatable_resource_ids = relatable_resource_ids + \
            [r2r.resourceclassto for r2r in Resource2ResourceConstraint.objects.filter(
                resourceclassfrom_id=self.nodeid)]

```

(continues on next page)

(continued from previous page)

```

    return relatable_resource_ids

    def set_relatable_resources(self, new_ids):
        old_ids = [res.nodeid for res in self.get_relatable_resources()]
        for old_id in old_ids:
            if old_id not in new_ids:
                Resource2ResourceConstraint.objects.filter(Q(resourceclassto_id=self.
↪nodeid) | Q(
                    resourceclassfrom_id=self.nodeid), Q(resourceclassto_id=old_id) |
↪Q(resourceclassfrom_id=old_id)).delete()
            for new_id in new_ids:
                if new_id not in old_ids:
                    new_r2r = Resource2ResourceConstraint.objects.create(
                        resourceclassfrom_id=self.nodeid, resourceclassto_id=new_id)
                    new_r2r.save()

    class Meta:
        managed = True
        db_table = 'nodes'

```

NodeGroup

```

class NodeGroup(models.Model):
    nodegroupid = models.UUIDField(primary_key=True, default=uuid.uuid1)
    legacygroupid = models.TextField(blank=True, null=True)
    cardinality = models.TextField(blank=True, default='1')
    parentnodegroup = models.ForeignKey('self', db_column='parentnodegroupid',
↪blank=True, null=True) # Allows nodegroups
↪within nodegroups

    class Meta:
        managed = True
        db_table = 'node_groups'

        default_permissions = ()
        permissions = (
            ('read_nodegroup', 'Read'),
            ('write_nodegroup', 'Create/Update'),
            ('delete_nodegroup', 'Delete'),
            ('no_access_to_nodegroup', 'No Access'),
        )

```

Edge

```
class Edge(models.Model):
    edgeid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This field type
    ↪ is a guess.
    name = models.TextField(blank=True, null=True)
    description = models.TextField(blank=True, null=True)
    ontologyproperty = models.TextField(blank=True, null=True)
    domainnode = models.ForeignKey('Node', db_column='domainnodeid', related_name='edge_
    ↪ domains')
    rangenode = models.ForeignKey('Node', db_column='rangenodeid', related_name='edge_
    ↪ ranges')
    graph = models.ForeignKey('GraphModel', db_column='graphid', blank=True, null=True)

    class Meta:
        managed = True
        db_table = 'edges'
        unique_together = (('rangenode', 'domainnode'),)
```

Function

```
class Function(models.Model):
    functionid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This field
    ↪ type is a guess.
    name = models.TextField(blank=True, null=True)
    functiontype = models.TextField(blank=True, null=True)
    description = models.TextField(blank=True, null=True)
    defaultconfig = JSONField(blank=True, null=True)
    modulename = models.TextField(blank=True, null=True)
    classname = models.TextField(blank=True, null=True)
    component = models.TextField(blank=True, null=True, unique=True)

    class Meta:
        managed = True
        db_table = 'functions'

    @property
    def defaultconfig_json(self):
        json_string = json.dumps(self.defaultconfig)
        return json_string

    def get_class_module(self):
        mod_path = self.modulename.replace('.py', '')
        module = None
        import_success = False
        import_error = None
        for function_dir in settings.FUNCTION_LOCATIONS:
            try:
                module = importlib.import_module(function_dir + '.%s' % mod_path)
                import_success = True
            except ImportError as e:
```

(continues on next page)

(continued from previous page)

```

        import_error = e
    if module != None:
        break
if import_success == False:
    print('Failed to import ' + mod_path)
    print(import_error)

func = getattr(module, self.classname)
return func

```

Ontologies

An ontology standardizes a set of valid CRM (Conceptual Reference Model) classes for Node instances, as well as a set of relationships that will define Edge instances. Most importantly, an ontology enforces which Edges can be used to connect which Nodes. If a pre-loaded ontology is designated for a Graph instance, every NodeGroup within that Graph must conform to that ontology. You may also create an “ontology-less” graph, which will not define specific CRM classes for the Nodes and Edges.

These rules are stored as *OntologyClass* instances, which are stored as JSON. These JSON objects consist of dictionaries with two properties, *down* and *up*, each of which contains another two properties *ontology_property* and *ontology_classes* (*down* assumes a known domain class, while *up* assumes a known range class).

```

{
  "down": [
    {
      "ontology_property": "P1_is_identified_by",
      "ontology_classes": [
        "E51_Contact_Point",
        "E75_Conceptual_Object_Appellation",
        "E42_Identifier",
        "E45_Address",
        "E41_Appellation"
      ]
    }
  ],
  "up": [
    {
      "ontology_property": "P1_identifies",
      "ontology_classes": [
        "E51_Contact_Point",
        "E75_Conceptual_Object_Appellation",
        "E42_Identifier"
      ]
    }
  ]
}

```

Arches comes preloaded with the [CIDOC CRM](#), an ontology created by ICOM (International Council of Museums) to model cultural heritage documentation. However, a developer may create and load an entirely new ontology.

Ontology

```
class Ontology(models.Model):
    ontologyid = models.UUIDField(default=uuid.uuid1, primary_key=True)
    name = models.TextField()
    version = models.TextField()
    path = models.FileField(storage=get_ontology_storage_system())
    parentontology = models.ForeignKey('Ontology', db_column='parentontologyid',
                                       related_name='extensions', null=True, blank=True)

    class Meta:
        managed = True
        db_table = 'ontologies'
```

OntologyClass

```
class OntologyClass(models.Model):
    """
    the target JSONField has this schema:

    values are dictionaries with 2 properties, 'down' and 'up' and within each of those
    ↪ another 2 properties,
    'ontology_property' and 'ontology_classes'

    "down" assumes a known domain class, while "up" assumes a known range class

    .. code-block:: python

        "down": [
            {
                "ontology_property": "P1_is_identified_by",
                "ontology_classes": [
                    "E51_Contact_Point",
                    "E75_Conceptual_Object_Appellation",
                    "E42_Identifier",
                    "E45_Address",
                    "E41_Appellation",
                    ....
                ]
            }
        ]
        "up": [
            {
                "ontology_property": "P1i_identifies",
                "ontology_classes": [
                    "E51_Contact_Point",
                    "E75_Conceptual_Object_Appellation",
                    "E42_Identifier"
                    ....
                ]
            }
        ]
```

(continues on next page)

(continued from previous page)

```

"""

ontologyclassid = models.UUIDField(default=uuid.uuid1, primary_key=True)
source = models.TextField()
target = JSONField(null=True)
ontology = models.ForeignKey('Ontology', db_column='ontologyid', related_name=
↳ 'ontologyclasses')

class Meta:
    managed = True
    db_table = 'ontologyclasses'
    unique_together = (('source', 'ontology'),)

```

RDM Models

The RDM (Reference Data Manager) stores all of the vocabularies used in your Arches installation. Whether they are simple wordlists or a polyhierarchical thesauri, these vocabularies are stored as “concept schemes” and can be viewed as an aggregation of one or more *concepts* and the semantic relationships (links) between those concepts.

In the data model, a concept scheme consists of a set of Concept instances, each paired with a *Value*. In our running name/name_type example, the Name Type.E55 Node would be linked to a Concept (Name Type.E55) which would have two child Concepts. Thus, where the user sees a dropdown containing “Primary” and “Alternate”, these are actually the Values of Name Type.E55’s two descendent Concepts. The parent/child relationships between Concepts are stored as *Relation* instances.

Concept

```

class Concept(models.Model):
    conceptid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This field_
↳ type is a guess.
    nodetype = models.ForeignKey('DNodeType', db_column='nodetype')
    legacyoid = models.TextField(unique=True)

    class Meta:
        managed = True
        db_table = 'concepts'

```

Relation

```

class Relation(models.Model):
    conceptfrom = models.ForeignKey(Concept, db_column='conceptidfrom', related_name=
↳ 'relation_concepts_from')
    conceptto = models.ForeignKey(Concept, db_column='conceptidto', related_name=
↳ 'relation_concepts_to')
    relationtype = models.ForeignKey(DRelationType, db_column='relationtype')
    relationid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This field_
↳ type is a guess.

```

(continues on next page)

(continued from previous page)

```
class Meta:
    managed = True
    db_table = 'relations'
    unique_together = (('conceptfrom', 'conceptto', 'relationtype'),)
```

Value

```
class Value(models.Model):
    valueid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This field type
    ↪ is a guess.
    concept = models.ForeignKey('Concept', db_column='conceptid')
    valuetype = models.ForeignKey(DValueType, db_column='valuetype')
    value = models.TextField()
    language = models.ForeignKey(DLanguage, db_column='languageid', blank=True,
    ↪ null=True)

    class Meta:
        managed = True
        db_table = 'values'
```

Resource Data

Three models are used to store Arches business data:

- ResourceInstance - one per resource in the database
- Tile - stores all business data
- ResourceXResource - records relationships between resource instances

Creating a new resource in the database instantiates a new *ResourceInstance*, which belongs to one resource model and has a unique `resourceinstanceid`. A resource instance may also have its own security/permissions properties in order to allow a fine-grained level of user-based permissions.

Once data have been captured, they are stored as Tiles in the database. Each Tile stores one instance of all of the attributes of a given NodeGroup for a resource instance, as referenced by the `resourceinstanceid`. This business data is stored as a JSON object, which is a dictionary with n number of keys/value pairs that represent a Node's id `nodeid` and that Node's value.

in theory:

```
{
    "nodeid": "node value",
    "nodeid": "node value"
}
```

in practice:

```
{
    "20000000-0000-0000-0000-000000000002": "John",
```

(continues on next page)

(continued from previous page)

```
}
    "200000000-0000-0000-0000-000000000004": "Primary"
```

(In keeping with our running example, the keys in the second example would refer to an `Name.E1` node and an `Name.Type.E55` node, respectively.)

Arches also allows for the creation of relationships between resource instances, and these are stored as instances of the `ResourceXResource` model. The `resourceinstanceidfrom` and `resourceinstanceidto` fields create the relationship, and `relationshiptype` qualifies the relationship. The latter must correspond to the appropriate top node in the RDM. This constrains the list of available types of relationships available between resource instances.

ResourceInstance

```
class ResourceInstance(models.Model):
    resourceinstanceid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This
    ↪field type is a guess.
    graph = models.ForeignKey(GraphModel, db_column='graphid')
    legacyid = models.TextField(blank=True, unique=True, null=True)
    createdtime = models.DateTimeField(auto_now_add=True)

    class Meta:
        managed = True
        db_table = 'resource_instances'
```

TileModel

```
class TileModel(models.Model): # Tile
    """
    the data JSONField has this schema:

    values are dictionaries with n number of keys that represent nodeid's and values the
    ↪value of that node instance

    .. code-block:: python

        {
            nodeid: node value,
            nodeid: node value,
            ...
        }

        {
            "200000000-0000-0000-0000-000000000002": "John",
            "200000000-0000-0000-0000-000000000003": "Smith",
            "200000000-0000-0000-0000-000000000004": "Primary"
        }

    the provisionaledits JSONField has this schema:
```

(continues on next page)

(continued from previous page)

values are dictionaries with n number of keys that represent nodeid's and values the
 ↳ value of that node instance

.. code-block:: python

```
{
    userid: {
        value: node value,
        status: "review", "approved", or "rejected"
        action: "create", "update", or "delete"
        reviewer: reviewer's user id,
        timestamp: time of last provisional change,
        reviewtimestamp: time of review
    }
    ...
}

{
    1: {
        "value": {
            "200000000-0000-0000-0000-00000000000002": "Jack",
            "200000000-0000-0000-0000-00000000000003": "Smith",
            "200000000-0000-0000-0000-00000000000004": "Primary"
        },
        "status": "rejected",
        "action": "update",
        "reviewer": 8,
        "timestamp": "20180101T1500",
        "reviewtimestamp": "20180102T0800",
    },
    15: {
        "value": {
            "200000000-0000-0000-0000-00000000000002": "John",
            "200000000-0000-0000-0000-00000000000003": "Smith",
            "200000000-0000-0000-0000-00000000000004": "Secondary"
        },
        "status": "review",
        "action": "update",
    }
}

"""

tileid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This field type
↳ is a guess.
resourceinstance = models.ForeignKey(ResourceInstance, db_column='resourceinstanceid
↳ ')
parenttile = models.ForeignKey('self', db_column='parenttileid', blank=True,
↳ null=True)
data = JSONField(blank=True, null=True, db_column='tiledata') # This field type is
↳ a guess.
nodegroup = models.ForeignKey(NodeGroup, db_column='nodegroupid')
sortorder = models.IntegerField(blank=True, null=True, default=0)
```

(continues on next page)

(continued from previous page)

```

provisionaledits = JSONField(blank=True, null=True, db_column='provisionaledits') #
↪ This field type is a guess.

class Meta:
    managed = True
    db_table = 'tiles'

def save(self, *args, **kwargs):
    if(self.sortorder is None or (self.provisionaledits is not None and self.data ==
↪ {})):
        sortorder_max = TileModel.objects.filter(
            nodegroup_id=self.nodegroup_id, resourceinstance_id=self.
↪ resourceinstance_id).aggregate(Max('sortorder'))['sortorder__max']
        self.sortorder = sortorder_max + 1 if sortorder_max is not None else 0
        super(TileModel, self).save(*args, **kwargs) # Call the "real" save() method.

```

ResourceXResource

```

class ResourceXResource(models.Model):
    resourceid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This field
↪ type is a guess.
    resourceinstanceidfrom = models.ForeignKey(
        'ResourceInstance', db_column='resourceinstanceidfrom', blank=True, null=True,
↪ related_name='resxres_resource_instance_ids_from')
    resourceinstanceidto = models.ForeignKey(
        'ResourceInstance', db_column='resourceinstanceidto', blank=True, null=True,
↪ related_name='resxres_resource_instance_ids_to')
    notes = models.TextField(blank=True, null=True)
    relationshiptype = models.TextField(blank=True, null=True)
    datestarted = models.DateField(blank=True, null=True)
    dateended = models.DateField(blank=True, null=True)
    created = models.DateTimeField()
    modified = models.DateTimeField()

def delete(self):
    from arches.app.search.search_engine_factory import SearchEngineFactory
    se = SearchEngineFactory().create()
    se.delete(index='resource_relations', doc_type='all', id=self.resourceid)
    super(ResourceXResource, self).delete()

def save(self):
    from arches.app.search.search_engine_factory import SearchEngineFactory
    se = SearchEngineFactory().create()
    if not self.created:
        self.created = datetime.datetime.now()
        self.modified = datetime.datetime.now()
        document = model_to_dict(self)
        se.index_data(index='resource_relations', doc_type='all', body=document, idfield=
↪ 'resourceid')
        super(ResourceXResource, self).save()

```

(continues on next page)

(continued from previous page)

```
class Meta:
    managed = True
    db_table = 'resource_x_resource'
```

Edit Log

A change in a Tile's contents, which is the result of any resource edits, is recorded as an instance of the EditLog model.

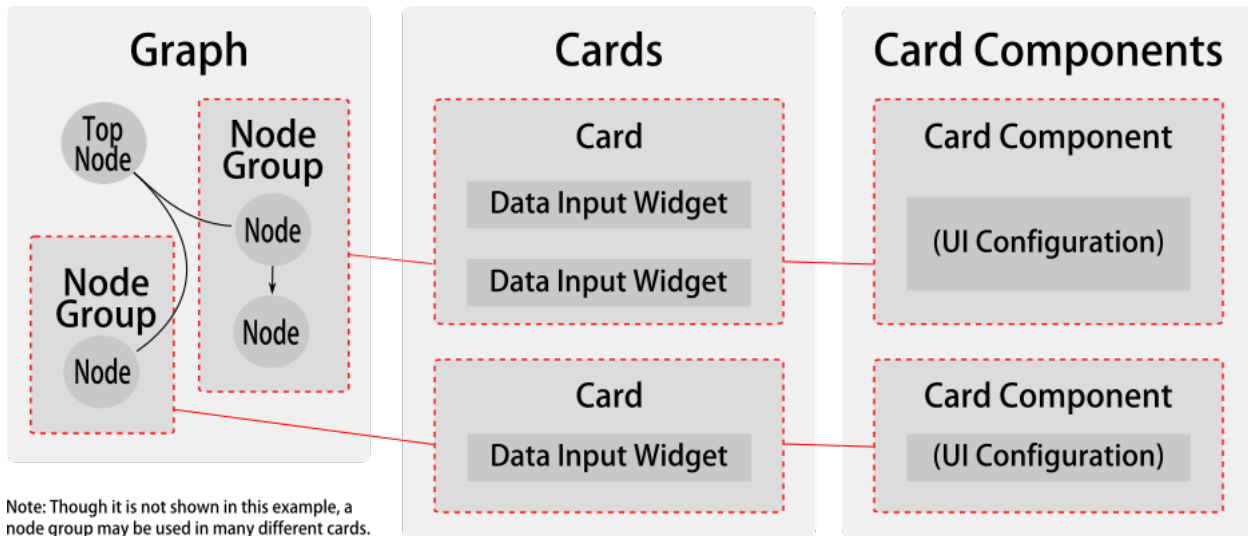
```
class EditLog(models.Model):
    editlogid = models.UUIDField(primary_key=True, default=uuid.uuid1)
    resourcedisplayname = models.TextField(blank=True, null=True)
    resourceclassid = models.TextField(blank=True, null=True)
    resourceinstanceid = models.TextField(blank=True, null=True)
    nodegroupid = models.TextField(blank=True, null=True)
    tileinstanceid = models.TextField(blank=True, null=True)
    edittype = models.TextField(blank=True, null=True)
    newvalue = JSONField(blank=True, null=True, db_column='newvalue')
    oldvalue = JSONField(blank=True, null=True, db_column='oldvalue')
    newprovisionalvalue = JSONField(blank=True, null=True, db_column='newprovisionalvalue')
    oldprovisionalvalue = JSONField(blank=True, null=True, db_column='oldprovisionalvalue')
    timestamp = models.DateTimeField(blank=True, null=True)
    userid = models.TextField(blank=True, null=True)
    user_firstname = models.TextField(blank=True, null=True)
    user_lastname = models.TextField(blank=True, null=True)
    user_email = models.TextField(blank=True, null=True)
    user_username = models.TextField(blank=True, null=True)
    provisional_userid = models.TextField(blank=True, null=True)
    provisional_user_username = models.TextField(blank=True, null=True)
    provisional_edittype = models.TextField(blank=True, null=True)
    note = models.TextField(blank=True, null=True)

class Meta:
    managed = True
    db_table = 'edit_log'
```

UI Component Models

A number of models exist specifically to support the resource model UI. The purpose of this is to create direct relationships between the resource graph and the data entry cards that are used to create resource instances. Generally, the process works like this:

1. A resource graph is an organized collection of NodeGroups which define what information will be gathered for a given resource model.
2. A resource's *Cards* are tied to specific NodeGroups and define which input *Widgets* will be used to gather values for each Node in that NodeGroup. *Card Components* are used to render the cards in various contexts in the Arches UI.



Cards are UI representations of a NodeGroup, and they encapsulate the Widgets that facilitate data entry for each Node in a given NodeGroup instance.

While a Card will only handle data entry for a single NodeGroup (which may have many Nodes or NodeGroups), a single NodeGroup can be handled by more than one Card.

Throughout the Arches UI, Card Components are used to render Cards in both read-only and data entry contexts.

Note: Beginning in Arches 4.3, Card Components provide functionality formerly provided by Forms, Menus, and Reports.

CardModel

```
class CardModel(models.Model):
    cardid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This field type
    ↪ is a guess.
    name = models.TextField(blank=True, null=True)
    description = models.TextField(blank=True, null=True)
    instructions = models.TextField(blank=True, null=True)
    cssclass = models.TextField(blank=True, null=True)
    helpenabled = models.BooleanField(default=False)
    helptitle = models.TextField(blank=True, null=True)
    helptext = models.TextField(blank=True, null=True)
    nodegroup = models.ForeignKey('NodeGroup', db_column='nodegroupid')
    graph = models.ForeignKey('GraphModel', db_column='graphid')
    active = models.BooleanField(default=True)
    visible = models.BooleanField(default=True)
    sortorder = models.IntegerField(blank=True, null=True, default=None)
    component = models.ForeignKey('CardComponent', db_column='componentid', default=uuid.
    ↪ UUID(
        'f05e4d3a-53c1-11e8-b0ea-784f435179ea'), on_delete=models.SET_DEFAULT)
    config = JSONField(blank=True, null=True, db_column='config')

    def is_editable(self):
```

(continues on next page)

(continued from previous page)

```
result = True
tiles = TileModel.objects.filter(nodegroup=self.nodegroup).count()
result = False if tiles > 0 else True
if settings.OVERRIDE_RESOURCE_MODEL_LOCK == True:
    result = True
return result

class Meta:
    managed = True
    db_table = 'cards'
```

Card Component

A Card Component renders a Card.

```
class CardComponent(models.Model):
    componentid = models.UUIDField(primary_key=True, default=uuid.uuid1)
    name = models.TextField(blank=True, null=True)
    description = models.TextField(blank=True, null=True)
    component = models.TextField()
    componentname = models.TextField()
    defaultconfig = JSONField(blank=True, null=True, db_column='defaultconfig')

    @property
    def defaultconfig_json(self):
        json_string = json.dumps(self.defaultconfig)
        return json_string

    class Meta:
        managed = True
        db_table = 'card_components'
```

Field description:

name

a name to be displayed in the UI for this component

description

a description to be displayed in the UI for this component

component

a require path for the JS module representing this component

componentname

a Knockout.js component name used by this component (for rendering via knockout's component binding handler)

defaultconfig

a default JSON configuration object to be used by cards that implement this component

Widget

```
class Widget(models.Model):
    widgetid = models.UUIDField(primary_key=True, default=uuid.uuid1) # This field type
    ↪is a guess.
    name = models.TextField(unique=True)
    component = models.TextField(unique=True)
    defaultconfig = JSONField(blank=True, null=True, db_column='defaultconfig')
    helptext = models.TextField(blank=True, null=True)
    datatype = models.TextField()

    @property
    def defaultconfig_json(self):
        json_string = json.dumps(self.defaultconfig)
        return json_string

    class Meta:
        managed = True
        db_table = 'widgets'
```

DDataType

Used to validate data entered into widgets

```
class DDataType(models.Model):
    datatype = models.TextField(primary_key=True)
    iconclass = models.TextField()
    modulename = models.TextField(blank=True, null=True)
    classname = models.TextField(blank=True, null=True)
    defaultwidget = models.ForeignKey(db_column='defaultwidget', to='models.Widget',
    ↪null=True)
    defaultconfig = JSONField(blank=True, null=True, db_column='defaultconfig')
    configcomponent = models.TextField(blank=True, null=True)
    configname = models.TextField(blank=True, null=True)
    issearchable = models.NullBooleanField(default=False)
    isgeometric = models.BooleanField()

    class Meta:
        managed = True
        db_table = 'd_data_types'
```

Naming Conventions

id vs _id: ID as Primary Key vs Foreign Key

Throughout the code, you will sometimes see an entity name with “id” appended and other times see the same name with “_id” appended. For example, you’ll see both `nodegroupid` and `nodegroup_id`.

What is the difference?

The first, `nodegroupid`, is a UUID attribute in the database and is the primary key for entities of type `NodeGroup`.

The second, `nodegroup_id`, is a foreign key attribute (thus also a UUID) that refers from somewhere else to a NodeGroup. For example, a Tile object may have an associated NodeGroup; that NodeGroup object itself would be referenced as `tile.nodegroup`, and the NodeGroup's UUID – which in the context of a Tile object is a foreign key – would therefore be `tile.nodegroup_id`.

The reason to use `tile.nodegroup_id`, instead of getting the NodeGroup's ID by going through the associated NodeGroup object with `tile.nodegroup.nodegroupid`, is that the latter would involve an extra database query to fetch the NodeGroup instance, which would be a waste if you don't actually need the NodeGroup itself. When all you need is the NodeGroup's UUID – perhaps because you're just going to pass it along to something else that only needs the UUID – then there's no point fetching the entire NodeGroup when you already have the Tile in hand and the Tile's `nodegroup_id` field is a foreign key to the Tile's associated NodeGroup. You might as well just get that foreign key, `tile.nodegroup_id`, directly.

Resource Import/Export

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

Importing Data

Arches provides methods for importing data in a few different formats. Generally, you are placing the values you want to import into a structured file. The form that each value takes, depends on the data type of its target node.

Be aware that the graph-based structure of Resource Models in Arches means that your data must be carefully prepared before import, to ensure that branches, groupings, and cardinality is maintained. The method for doing this is determined by which file format you decide to use. Additionally, the data type of the target node for each value in your file will dictate that value's format.

Datatype Formats

Nodes in your target resource model will have a specific datatype defined for each one (see [Core Arches Datatypes](#)), and it is very important that you format your input data accordingly. Below is a list of all core datatypes and how they should look in your import files.

string

Strings can be simple text or include HTML tags (whether or not HTML is rendered depends on the card and widget configuration):

```
Smith Cottage
<p>This is a rich text description that contains <strong>HTML</strong> tags.</p>
```

In CSV, strings must be quoted only if they contain a comma:

```
"Behold, the Forevertron."
```


number

Integers or floats; never use quotes or comma separators:

```
42
-465
17322.464453
```

date

Format must be YYYY-MM-DD, no quotes:

```
1305-10-31
1986-02-02
```

edtf

Must be a valid [Extended Date Time Format](#) string:

```
"2010-10"
"-y10000"
```

Arches supports level 2 of the EDTF specification. However, because of a bug in the edtf package used by Arches, an error will be thrown for strings like:

```
"../1924"
```

As a workaround, you can use a string like:

```
"[../1924]"
```

geojson-feature-collection

In CSV, use the [Well-Known Text \(WKT\)](#) format:

```
POINT (-82.53973 29.658642)
MULTIPOLYGON (((-81.435 26.130, -81.425 26.124, -81.415 26.137, -81.435 26.130)))
```

In JSON, include the entire definition of a [GeoJSON Feature Collection](#) (the `properties` and `id` attributes can be empty). Use geojson.io and geojsonlint.com for testing:

```
"features": [
  {
    "geometry": {
      "coordinates": [
        -82.53973,
        29.658642
      ],
      "type": "Point"
    },
  },
]
```

(continues on next page)

(continued from previous page)

```
        "id": "",
        "properties": {},
        "type": "Feature"
      }
    ],
    "type": "FeatureCollection"
  }
```

concept

In CSV/SHP, if the values in your concept collection are *unique* you can use the label (prefLabel) for a concept. If not, you will get an error during import and you must use UUIDs instead of labels (if this happens, see [Concepts File](#) below):

```
Slate
2995daea-d6d3-11e8-9eb1-0242ac150004
```

If a prefLabel has a comma in it, it must be **triple-quoted**:

```
""""Shingles, original""""
```

In JSON, you must use a concept's UUID:

```
2995daea-d6d3-11e8-9eb1-0242ac150004
```

concept-list

In CSV/SHP, must be a single-quoted list of prefLabels (or UUIDs if necessary):

```
Brick
"Slate,Thatch"
"651c59b0-ff30-11e8-9975-94659cf754d0,cdcc206d-f80d-4cc3-8685-40e8949158f8"
```

If a prefLabel contains a comma, then that prefLabel must be **double-quoted**:

```
"Slate,""Shingles, original"",Thatch"
```

In JSON, a list of UUIDs must be used. If only one value is present, it must still be placed within brackets:

```
["d11630fa-c5a4-49b8-832c-5976e0044bca"]
["651c59b0-ff30-11e8-9975-94659cf754d0","cdcc206d-f80d-4cc3-8685-40e8949158f8"]
```

domain-value

A string that matches a valid domain value for this node, single-quoted if it contains a comma:

```
Yes
"Started, in progress"
```

domain-value-list

A single-quoted list of strings that match valid domain values for this node. Follow quoting guidelines for *concept-list* if any values contain commas:

```
"Red,Blue,Green"
```

file-list

In CSV/SHP, simply use the file name, or a single-quoted list of file names:

```
BuildingPicture.jpg
```

See the note below about where to prepopulate this file on your server, if you are not uploading it through the package load operation.

In JSON, you must include a more robust definition of the file that looks like this (and remember, this must be a list, even if you only have one file per node):

```
[
  {
    "accepted": true,
    "file_id": "6304033b-2f42-4bfd-86a5-5e2a941d95f1",
    "name": "BuildingPicture.jpg",
    "renderer": "5e05aa2e-5db0-4922-8938-b4d2b7919733",
    "status": "uploaded",
    "type": "image/jpeg",
    "url": "/files/6304033b-2f42-4bfd-86a5-5e2a941d95f1"
  }
]
```

You should be able to generate this content by doing the following:

1. Pregenerate a new UUID for each file
2. Place this UUID in the `file_id` property, and also use it in the `url` property as shown above.
3. Select a renderer from `settings.RENDERERS` (see [settings.py](#)) and use its id for the `renderer` property. At the time of this writing, use `5e05aa2e-5db0-4922-8938-b4d2b7919733` for images (jpg, png, etc.) and `09dec059-1ee8-4fbd-85dd-c0ab0428aa94` for PDFs.
4. Set the `type` as appropriate—`image/jpeg`, `image/png`, `application/pdf`, etc.

Note: The file(s) should already exist in the `uploadedfiles/` directory prior to loading the resource, but technically can be added later as well. This directory should be located *within* your `MEDIA_ROOT` location. For example, by

default, Arches sets `MEDIA_ROOT = os.path.join(ROOT_DIR)`. This means you should find (or create if it doesn't exist) `my_project/uploadedfiles`, alongside `manage.py`.

resource-instance

In CSV/SHP, the format consists of a version of the JSON data structure:

```
"[{'resourceId': '3d5a80df-4bcb-4ea0-bbaf-327ea0f41b31', 'ontologyProperty': 'http://www.
↳cidoc-crm.org/cidoc-crm/L54i_is_same-as', 'resourceXresourceId': '',
↳'inverseOntologyProperty': 'http://www.cidoc-crm.org/cidoc-crm/L54i_is_same-as'}]"
```

Where:

- **resourceId** (*required*) - the target resource-instance ResourceID
- **ontologyProperty** (*can be left blank*) - the URL of the ontology property that defines the relationship to the target resource-instance
- **resourceXresourceId** (*can be left blank*) - the system will assign a UUID for this relationship
- **inverseOntologyProperty** (*can be left blank*) - the URL of the ontology property that defines the inverse of the relationship referenced under **ontologyProperty**

In JSON, the format is as follows:

```
{
  "inverseOntologyProperty": "",
  "ontologyProperty": "",
  "resourceId": "b2f2f91f-2881-11ed-ad39-e746f226a47a",
  "resourceXresourceId": ""
}
```

resource-instance-list

In CSV/SHP, same as above, except repeating each resource-instance within the square brackets (i.e. “[{first resource-instance},{second resource-instance}]”):

```
"[{'resourceId': '3d5a80df-4bcb-4ea0-bbaf-327ea0f41b31', 'ontologyProperty': 'http://www.
↳cidoc-crm.org/cidoc-crm/L54i_is_same-as', 'resourceXresourceId': '',
↳'inverseOntologyProperty': 'http://www.cidoc-crm.org/cidoc-crm/L54i_is_same-as'},{
↳'resourceId': 'celefa88-d68e-44e3-95fa-3abb2cb433e9', 'ontologyProperty': 'http://www.
↳cidoc-crm.org/cidoc-crm/L54i_is_same-as', 'resourceXresourceId': '',
↳'inverseOntologyProperty': 'http://www.cidoc-crm.org/cidoc-crm/L54i_is_same-as'}]"
```

In JSON:

```
[
  {
    "inverseOntologyProperty": "",
    "ontologyProperty": "",
    "resourceId": "b2f2f91f-2881-11ed-ad39-e746f226a47a",
    "resourceXresourceId": ""
  },

```

(continues on next page)

(continued from previous page)

```
{
  "inverseOntologyProperty": "",
  "ontologyProperty": "",
  "resourceId": "b94455a2-a8ed-4d3d-919a-ae91493d6606",
  "resourceXresourceId": ""
}
```

url

Same as *string* formatting. Validation will run to ensure the value is a proper URL:

```
https://www.nps.gov/subjects/nationalregister/index.htm
```

CSV Import

One method of bulk loading data into Arches is to create a CSV (comma separated values) file. We recommend using MS Excel or Open Office for this task. More advanced users will likely find a custom scripting effort to be worthwhile.

Note: Your CSV should be encoded into UTF-8. [These steps](#) will help you if you are using MS Excel.

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.

ResourceID	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.

ResourceID	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).

ResourceID	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. Note that “Primary” and “Historic” are the prefLabels for two different concepts in the RDM.

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

Note: If you are using MS Excel to create your CSV files, double-quotes will automatically be added to any cell value that contains a comma.

Mapping File

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 (see [Concepts File](#)). 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 compose 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 specific node simply set the `file_field_name` attribute to `""`.

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.

Concepts File

When populating concept nodes from a CSV you should generally use the prefLabel for that concept. However, in rare instances there may be two or more concepts in your collection that have identical prefLabels (this is allowed in Arches). In this case you will need to replace the prefLabel in your CSV with the UUID for the Value that represents that prefLabel.

To aid with the process, a “concepts file” is created every time you download a mapping file, which lists the valueids and corresponding labels for all of the concepts in 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"  
},
```

You would then need to use 81dd62d2-6701-4195-b74b-8057456bba4b instead of Primary in your CSV.

Shapefile Import

```
python manage.py packages -o import_business_data -s 'path_to_shapefile' -c 'path_to_  
↪mapping_file' [-ow {'overwrite'|'append'}]
```

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. Note that in this mapping file, the node you wish to map the geometry to must have a `file_field_name` value of ‘geom’.

Other Requirements:

- 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. They can also be strings stored in YYYY-MM-DD format.

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.

JSON Import

```
python manage.py packages -o import_business_data -s 'path_to_json' [-ow {'overwrite'|
↪ 'append'}]
```

JSON import of business data is primarily intended for transferring business data between arches instances. Because of this it's not especially user-friendly to create or interpret the JSON data format, but doing so is not impossible.

First, there are at least two ways you can familiarize yourself with the format. The system settings in an Arches package is stored in this json format, you can open one of those up and take a look. Perhaps a better way in your case is to create some business data via the ui in your instance of arches and export it to the json format using the business data export command defined here [Export Commands](#). This can act as a template json for data creation. For the rest of this section it may be helpful to have one of these files open to make it easier to follow along.

General structure of the entire file:

```
{
  "business_data": {
    "resources": [
      {
        "resourceinstance": { . . . },
        "tiles": [ . . . ],
      }
    ]
  }
}
```

The json format is primarily a representation of the tiles table in the arches postgres database with some information about the resource instance(s) included. Within the business_data object of the json are two objects, the tiles object and the resourceinstance object. Let's start with the resource instance object.

Structure of the resourceinstance object:

```
{
  "graph_id": uuid,
  "resourceinstanceid": uuid,
  "legacyid": uuid or text
}
```

- **graph_id** - the id of the resource model for which this data was created
- **resourceinstanceid** - the unique identifier of this resource instance within Arches (this will need to be unique for every resource in Arches)
- **legacyid** - an identifier that was used for this resource before its inclusion in Arches. This can be the same as the resourceinstanceid (this is the case when you provide a UUID to the ResourceID column in a CSV) or it can be another id. Either way it has to be unique among every resource in Arches.

The tiles object is a list of tiles that compose a resource instance. The tiles object is a bit more complicated than the resourceinstance object, and the structure can vary depending on the cardinality of your nodes. The following cardinality examples will be covered below:

1. *1 card*
2. *n cards*
3. *1 parent card with 1 child card*
4. *1 parent card with n child cards*

5. *n* parent cards with 1 child card
6. *n* parent cards with *n* child cards

But first a description of the general structure of a single tile:

```
{
  "tileid": "<uuid>",
  "resourceinstance_id": "<uuid>",
  "nodegroup_id": "<uuid>",
  "sortorder": 0,
  "parenttile_id": "<uuid>" or null,
  "data": { . . . }
}
```

- **tileid** - unique identifier of the tile this is the primary key in the tiles table and must be a unique uuid
- **resourceinstance_id** - the uuid corresponding to the instance this tile belongs to (this should be the same as the resourceinstance_id from the resourceinstance object.
- **nodegroup_id** - the node group for which the nodes within the data array participate
- **sortorder** - the sort order of this data in the form/report relative to other tiles (only applicable if cardinality is *n*)
- **parenttile_id** - unique identifier of the parenttile of this tile (will be null if this is a parent tile or the tile has no parent)
- **data** - json structure of a node group including the nodeid and data populating that node. For example:

```
{
  "data": {
    "<uuid for building name node>": "Smith Cottage"
  }
}
```

The tile object is tied to a resource model in two ways: 1) through the nodegroup_id 2) in the data object where nodeids are used as keys for the business data itself.

Now for a detailed look at the actual contents of tiles. Note that below we are using simplified values for tileid, like "A" and "B", to clearly illustrate parent/child relationships. In reality these must be valid UUIDs.

1 card

1: There is one and only one instance of this nodegroup/card in a resource:

```
[
  {
    "tileid": "A",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": null,
    "data": {
      "nodeid": "some data",
      "nodeid": "some other data"
    }
  }
]
```

(continues on next page)

(continued from previous page)

```
}
]
```

This structure represents a tile for a nodegroup (consisting of two nodes) with no parents collecting data with a cardinality of 1.

n cards

n: There are multiple instances of this nodegroup/card in a resource:

```
[
  {
    "tileid": "A",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": null,
    "data": {
      "nodeid": "some data",
      "nodeid": "some other data"
    }
  },
  {
    "tileid": "B",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": null,
    "data": {
      "nodeid": "more data",
      "nodeid": "more other data"
    }
  }
]
```

1 parent card with 1 child card

1-1: One and only one parent nodegroup/card contains one and only one child nodegroup/card:

```
[
  {
    "tileid": "A",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": null,
    "data": {}
  },
  {
    "tileid": "X",
```

(continues on next page)

(continued from previous page)

```
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": "A",
    "data": {
        "nodeid": "data",
        "nodeid": "other data"
    }
}
]
```

1 parent card with n child cards

1-n: One and only one parent nodegroup/card containing multiple instances of child nodegroups/cards:

```
[
  {
    "tileid": "A",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": null,
    "data": {}
  },
  {
    "tileid": "X",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": "A",
    "data": {
        "nodeid": "data",
        "nodeid": "other data"
    }
  },
  {
    "tileid": "Y",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": "A",
    "data": {
        "nodeid": "more data",
        "nodeid": "more other data"
    }
  }
]
```

n parent cards with 1 child card

n-1: Many parent nodegroups/cards each with one child nodegroup/card:

```
[
  {
    "tileid": "A",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": null,
    "data": {}
  },
  {
    "tileid": "X",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": "A",
    "data": {
      "nodeid": "data",
      "nodeid": "other data"
    }
  },
  {
    "tileid": "B",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": null,
    "data": {}
  },
  {
    "tileid": "Y",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": "B",
    "data": {
      "nodeid": "more data",
      "nodeid": "more other data"
    }
  }
]
```

n parent cards with n child cards

n-n: Many parent nodegroups/cards containing many child nodegroups/cards:

```
[
  {
    "tileid": "A",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": null,
    "data": {}
  },
  {
    "tileid": "X",
    "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
    "nodegroup_id": "<uuid from resource model>",
    "sortorder": 0,
    "parenttile_id": "A",
    "data": {
      "nodeid": "data",
      "nodeid": "other data"
    }
  },
  {
    {
      "tileid": "B",
      "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
      "nodegroup_id": "<uuid from resource model>",
      "sortorder": 0,
      "parenttile_id": null,
      "data": {}
    },
    {
      "tileid": "Y",
      "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
      "nodegroup_id": "<uuid from resource model>",
      "sortorder": 0,
      "parenttile_id": "B",
      "data": {
        "nodeid": "more data",
        "nodeid": "more other data"
      }
    }
  },
  {
    {
      "tileid": "Z",
      "resourceinstance_id": "<uuid from resourceinstance.resourceinstanceid>",
      "nodegroup_id": "<uuid from resource model>",
      "sortorder": 0,
      "parenttile_id": "B",
      "data": {
        "nodeid": "even more data",
        "nodeid": "even more other data"
      }
    }
  }
]
```

(continues on next page)

(continued from previous page)

```
}
]
```

Importing Resource Relations

It is possible to batch import Resource Relations (also referred to as “resource-to-resource relationships”). To do so, create a `.relations` file (a CSV-formatted file with a `.relations` extension). The header of the file should be as follows:

```
resourceinstanceidfrom,resourceinstanceidto,relationshiptype,datestarted,dateended,notes
```

In each row, `resourceinstanceidfrom` and `resourceinstanceidto` must either be an Arches ID (the UUID assigned to a new resource when it is first created) or a Legacy ID (an identifier from a legacy database that was used as a `ResourceID` in a JSON or CSV import file).

You can find the UUID value for your desired `relationshiptype` in the `concept.json` file downloaded with your resource model mapping file.

`datestarted`, `dateended` and `notes` are optional fields. Dates should be formatted YYYY-MM-DD.

Once constructed you can import the `.relations` file with the following command:

```
python manage.py packages -o import_business_data_relations -s 'path_to_relations_file'
```

All the resources referenced in the `.relations` CSV need to already be in your database. So make sure to run this command *after* you have imported all the business data referenced in the `.relations` file.

Note: You can also create relationships between resources using the `resource-instance` data type. When you are making the graph for a new resource model, you can set one of the nodes to hold a resource instance. This is not the same as creating Resource Relations as described above.

SQL Import

Arches provides database functions that are meant to assist with the loading, updating and querying of Arches business data via SQL. This strategy is especially useful if you are migrating an existing SQL database into Arches.

SQL import is more flexible and faster than loading via CSV, however it requires some SQL skills to write scripts to interact with these data.

The core functions that arches provides allow for flexible, on-demand creation of view entities that create relational database entities representing Arches graph schema in the form of database views. These database views can be queried using SQL, including *INSERT*, *UPDATE*, and *DELETE* operations.

View creation functions

`__arches_create_nodegroup_view`

Creates a view representing a specific nodegroup in the Arches graph schema. The resultant view can be queried using SQL including *INSERT*, *UPDATE*, and *DELETE* operations. If no view name is provided, then the function will attempt to create a view with the name of the nodegroup's root node processed to be suitable for a database entity name (for example, spaces replaced with underscores).

Arguments

group_id

uuid - the UUID of the nodegroup for which a view will be created.

view_name

text (optional) - the name to be used for the view being created, defaults to null

schema_name

text (optional) - the name of the schema to which the new view will be added, defaults to 'public'

parent_name

text (optional) - name used for column containing the parent tile id, defaults to 'parenttileid'

Returns

returns

text - message indicating success and name of the view created.

`__arches_create_branch_views`

Creates a series of views (using the above `__arches_create_nodegroup_view` function) representing a specific nodegroup and all of its child nodegroups (recursively) in the Arches graph schema.

Arguments

group_id

uuid - the UUID of the nodegroup for which views will be created recursively.

schema_name

text (optional) - the name of the schema to which the new views will be added, defaults to 'public'

Returns

returns

text - message indicating success.

`__arches_create_resource_model_views`

(Drops if it exists and) creates a schema and a view representing the instances of a specific resource model and series of views (using the above `__arches_create_nodegroup_view` function) for each of its nodegroups in the Arches graph schema. If no schema name is provided, then the function will attempt to create a schema with the name of resource model processed to be suitable for a database entity name (for example, spaces replaced with underscores).

Arguments

model_id

uuid - the UUID of the resource model for which views will be created.

schema_name

text (optional) - the name of the schema to which the new views will be added, defaults to null

Returns**returns**

text - message indicating success and the name of the schema created.

Helper functions

In addition to the functions that create views, the helper functions are also available to assist in the creation of tile data using the created views.

`__arches_get_node_id_for_view_column`

Returns the node id for a given view column. This is useful for subsequently looking up additional information about a column/node in the Arches graph schema, for example, creating a lookup table of concepts for a particular column/node.

Arguments**schema_name**

text - the name of the schema that contains the view of interest

view_name

text - the name of the view of interest

column_name

text - the name of the column of interest

Returns**returns**

uuid - the node id for the column of interest

`__arches_get_labels_for_concept_node`

Creates a lookup table of concepts for a particular column/node.

Arguments**node_id**

uuid - the UUID for a node in the Arches graph schema for which a lookup table of concepts will be created

language_id

text (optional) - the language id for which to return a lookup of concept values, defaults to 'en'

Returns**returns**

table - the lookup table of concepts for the column/node of interest. the resultant table's schema is:

depth

int - the depth of the concept values in the concept hierarchy

valueid

uuid - the value record's primary key

value

text - the value itself (the concept's label)

conceptid

uuid - the concept record's primary key

Example Usage

For a hypothetical example, consider a table in your legacy database called `buildings` with a `name` and `resourceid` columns. The following could be used to migrate the rows into new Arches resource instances.

Let's assume we have a Resource Model called "Architectural Resource", and it has two nodes, "Name" and "Name Type", under a single semantic node "Names".

Use the `__arches_create_resource_model_views` function (see above) to create a new schema for each active Resource Model.

```
SELECT __arches_create_resource_model_views(graphid) FROM graphs
WHERE isactive = true
AND name != 'Arches System Settings';
```

In our case, the result will be a new schema called `architectural_resource` and a table called `names` (named for the node furthest up the hierarchy in the nodegroup, in this case, a semantic node).

Directly inserting our records into the new Arches view will look something like this:

```
INSERT INTO architectural_resource.names (
    name,
    name_type,
    resourceinstanceid,
    transactionid
) select
    name,
    "Primary",
    resourceid,
    transactionid
from legacy_db.buildings;
```

Note: In this case, "Primary" is being given to every name type, because your legacy database did not have more than one name per resource.

Todo: A second table may need to be populated here too, to register the instances themselves.

Warning: This SQL method for inserting records has a known and severe performance issue for Postgres/PostGIS instances installed on Ubuntu, Debian, and Alpine operating systems. On these operating systems, a node-instance data insert of only a few thousand records may result in a database connection time out error (see issue discussion here: <https://github.com/archesproject/arches/issues/9049#issuecomment-1433970369>).

This issue is known to impact Arches versions 7.x. A fix for this OS related issue will likely come with Arches version 7.5. If you are using a version of Arches impacted by this issue, you can use the following workaround to

vastly (perhaps 50x) improve the performance of the SQL method for inserts. Execute the following SQL *BEFORE* you run SQL inserts:

```
create or replace function __arches_tile_view_update() returns trigger as $$
declare
    view_namespace text;
    group_id uuid;
    graph_id uuid;
    parent_id uuid;
    tile_id uuid;
    transaction_id uuid;
    json_data json;
    old_json_data jsonb;
    edit_type text;
begin
    select graphid into graph_id from nodes where nodeid = group_id;
    view_namespace = format('%s.%s', tg_table_schema, tg_table_name);
    select obj_description(view_namespace::regclass, 'pg_class') into group_id;
    if (TG_OP = 'DELETE') then
        select tiledata into old_json_data from tiles where tileid = old.tileid;
        delete from resource_x_resource where tileid = old.tileid;
        delete from public.tiles where tileid = old.tileid;
        insert into bulk_index_queue (resourceinstanceid, createddate)
            values (old.resourceinstanceid, current_timestamp) on conflict do nothing;
        insert into edit_log (
            resourceclassid,
            resourceinstanceid,
            nodegroupid,
            tileinstanceid,
            edittype,
            oldvalue,
            timestamp,
            note,
            transactionid
        ) values (
            graph_id,
            old.resourceinstanceid,
            group_id,
            old.tileid,
            'tile delete',
            old_json_data,
            now(),
            'loaded via SQL backend',
            public.uuid_generate_v1mc()
        );
        return old;
    else
        select __arches_get_json_data_for_view(new, tg_table_schema, tg_table_name)
        ↪into json_data;
        select __arches_get_parent_id_for_view(new, tg_table_schema, tg_table_name)
        ↪into parent_id;
        tile_id = new.tileid;
        if (new.transactionid is null) then
            transaction_id = public.uuid_generate_v1mc();
```

```
else
    transaction_id = new.transactionid;
end if;

if (TG_OP = 'UPDATE') then
    select tiledata into old_json_data from tiles where tileid = tile_id;
    edit_type = 'tile edit';
    if (transaction_id = old.transactionid) then
        transaction_id = public.uuid_generate_v1mc();
    end if;
    update public.tiles
    set tiledata = json_data,
        nodegroupid = group_id,
        parenttileid = parent_id,
        resourceinstanceid = new.resourceinstanceid
    where tileid = new.tileid;
elsif (TG_OP = 'INSERT') then
    old_json_data = null;
    edit_type = 'tile create';
    if tile_id is null then
        tile_id = public.uuid_generate_v1mc();
    end if;
    insert into public.tiles(
        tileid,
        tiledata,
        nodegroupid,
        parenttileid,
        resourceinstanceid
    ) values (
        tile_id,
        json_data,
        group_id,
        parent_id,
        new.resourceinstanceid
    );
end if;
perform __arches_refresh_tile_resource_relationships(tile_id);
insert into bulk_index_queue (resourceinstanceid, createddate)
    values (new.resourceinstanceid, current_timestamp) on conflict do nothing;
insert into edit_log (
    resourceclassid,
    resourceinstanceid,
    nodegroupid,
    tileinstanceid,
    edittype,
    newvalue,
    oldvalue,
    timestamp,
    note,
    transactionid
) values (
    graph_id,
```

```

        new.resourceinstanceid,
        group_id,
        tile_id,
        edit_type,
        json_data::jsonb,
        old_json_data,
        now(),
        'loaded via SQL backend',
        transaction_id
    );
    return new;
end if;
end;
$$ language plpgsql;

```

As part of this workaround, *after* you make any bulk updates or inserts to geometries, you'll need execute the following:

```
select * from refresh_geojson_geometries();
```

Exporting Arches Data

All file-based business exports must happen through the command line interface. The output format can either be JSON (the best way to do a full dump of your Arches database) or CSV (a more curated way to export a specific subset of data). To use Arches data in other systems or export shapefiles, users will have to begin by creating a new resource database view (see below).

Writing Business Data Files

The output format can either be JSON (the best way to do a full dump of your Arches database) or CSV (a more curated way to export a specific subset of data).

To export JSON, use:

```
python manage.py packages -o export_business_data -d 'path_to_destination_directory' -f
↪ 'json' -g 'resource_model_uuid'
```

Note that you'll have to provide the UUID for the Resource Model whose resources you want to export. The easiest way to find this UUID is by looking at the browser url while editing the Resource Model in the Arches Designer UI.

To export CSV, use:

```
python manage.py packages -o export_business_data -d 'path_to_destination_directory' -f
↪ 'csv' -c 'path_to_mapping_file' -g 'resource_model_uuid'
```

When exporting to CSV, you need to use a *Mapping File*, which will determine the content of your CSV (which nodes are exported, etc.). Add the `--single_file` argument to export your grouped data to the same CSV file as the rest of your data.

More about these export commands can be found in *Export Commands*.

Resource Database Views

To export to spatial 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 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,
```

(continues on next page)

(continued from previous page)

```

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'

```

- 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'
!= ''

```

- 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

```

- 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
```

Creating Applications

Starting with version 7.5, Arches moved to a new architectural pattern to support certain customization needs. This new pattern called **Arches Applications** (alternatively **Arches Apps**) should make customizations easier to develop and maintain. This architectural pattern also aligns with standard Django practices for the introduction of reusable sets of new features.

What's an App?

The phrase **Arches application** (or **Arches app**) describes a Python package that provides some set of features added to the core (standard) Arches application. Arches apps can be reused in multiple Arches projects. This terminology about *applications* and *projects* purposefully aligns with the [Django definition and use of these terms](#).

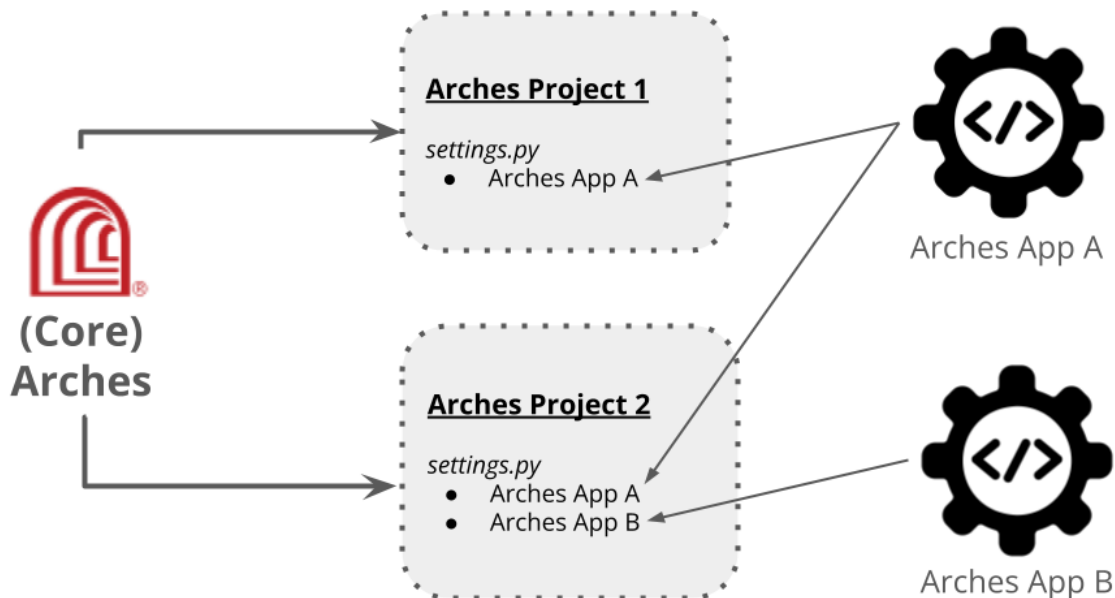


Fig. 34: Illustration of Arches projects integrating custom Arches Apps.

When are Arches Apps Useful?

Arches Apps provide a means to power special purpose features that may not be appropriate for incorporation into the core (standard) Arches application. A given Arches App can be under version control independent of core Arches. This should make it easier to update and upgrade core Arches independently of a custom Arches App (and vice versa).

A given Arches App can also be developed and shared open source. This means that the custom features powered by an Arches App can be reused widely across the community. Because Arches App development can proceed independently of core Arches, Arches Apps can be an excellent way for community members to experiment with features beyond those listed on the official Arches software development roadmap [official Arches software development roadmap](#).

[Arches for Science](#) illustrates the value of Arches apps. Arches for Science has several workflows and features (together with additional software dependencies) useful for cultural heritage conservation science. However, these features would be unnecessary for many other core Arches use cases. Keeping these conservation science features in a distinct app allows [Arches for Science software development](#) to continue at its own pace, and it reduces pressures to add highly specialized features to core Arches. Arches apps can therefore help reduce the complexity and maintenance costs of core Arches.

Arches Apps Can Help Avoid Forks

Through Arches apps, desired special features can be added to an Arches instance without forking the core (standard) Arches application code. There are many advantages to avoiding forks of the core (standard) Arches application code. By avoiding forks, one can more easily take advantage of continued upgrades and security patches applied to core Arches. This makes your use of Arches easier to maintain and secure.

A given Arches App can also be developed and shared open source. This means that the custom features powered by an Arches App can be reused across the community in multiple Arches projects.

Getting Started with Arches Apps

The Arches team created a simple example Arches app to illustrate how to develop and deploy custom apps. The example app called **Arches Dashboard** displays a summary count of resource instances and tiles in a given Arches project.

The **Arches Dashboard** app provides an example of how to build a custom Arches application. Experience with Django in general, and [Django app development](#) in particular, would be very useful for Arches app development. The official Django documentation provides a great starting [tutorial for learning how to create apps](#).

Installing the Arches Dashboard App

You can add the dashboard to an Arches project in just a few easy steps.

1. Install it from this repo (or clone this repo and pip install it locally).

```
pip install git+https://github.com/chiatt/dashboard.git
```

2. Add 'dashboard' as to the ARCHES_APPLICATIONS and INSTALLED_APPS settings in the demo project's settings.py file

```
ARCHES_APPLICATIONS = ("dashboard",) # be sure to add the trailing comma!
INSTALLED_APPS = [
    ...
    "demo",
```

(continues on next page)

(continued from previous page)

```
"dashboard",  
]
```

3. Update your `urls.py` file in your project. You'll likely need to add the `re_path` import:

```
from django.urls import include, path, re_path
```

and then the following path:

```
re_path(r"^", include("dashboard.urls")),
```

4. From your project run `migrate` to add the model included in the app:

```
python manage.py migrate
```

5. Next be sure to rebuild your project's frontend to include the plugin:

```
yarn build_development
```

6. When you're done you should see the Dashboard plugin added to you main navigation bar:

Creating Extensions

There are a number of patterns in place to allow you to extend Arches. Extensions can be used to customize the data entry process, add custom display widgets to reports, or even define new types of data that Arches can store.

Types of Extensions

Card Components

Beginning in Arches 4.3, Cards are rendered using Card Components, allowing them to be composed and nested arbitrarily in various contexts within the Arches UI. Arches comes with a default Card Component that should suit most needs, but you can also create and register custom Card Components to extend the front-end behavior of Arches.

Before exploring how do make customized Cards, please review documentation about available *Card Types* standard with Arches.

Developing Card Components is very similar to developing Widgets. A Card Component consists of a Django template and Knockout.js JavaScript file. To register your component, you'll also need a JSON file specifying its initial configuration.

To develop your new card, you'll place files like so in your project:

```
project_name/templates/views/components/cards/my-new-card.htm      project_name/  
media/js/views/components/cards/my-new-card.js
```

To register and configure the Component, you'll need a JSON configuration file:

```
project_name/cards/my-new-card.json
```

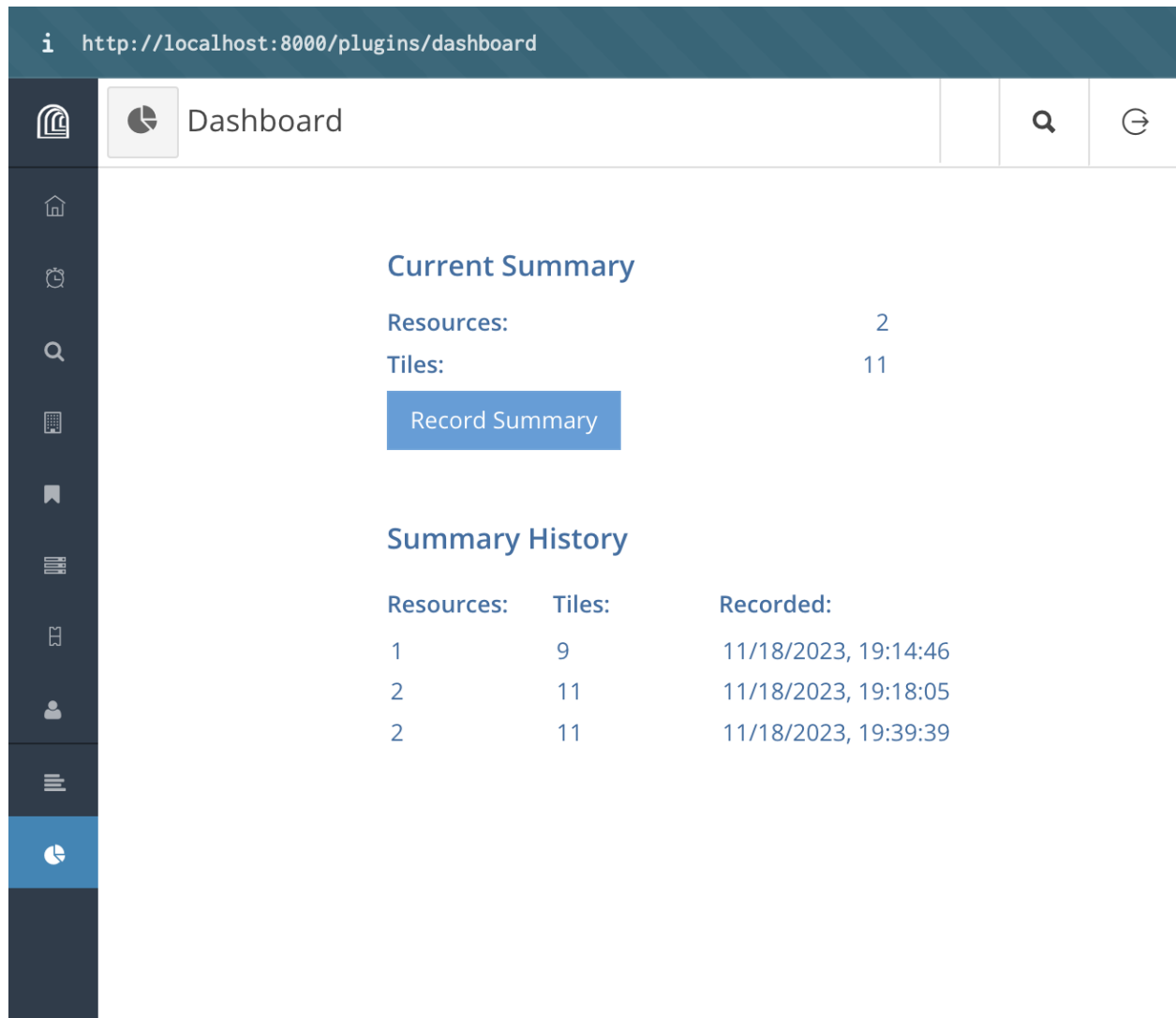


Fig. 35: A screenshot of the functioning **Arches Dashboard** app.

Creating a Card Component

The default template and Knockout files illustrate everything a Card Component needs, and you'll be extending this functionality. Your template will provide conditional markup for various contexts ('editor-tree', 'designer-tree', 'permissions-tree', 'form', and 'report'), render all the card's Widgets, and display other information.

Here's the template for the default Card Component:

```
{% load i18n %}
<!-- ko foreach: { data: [$data], as: 'self' } -->

<!-- ko if: state === 'editor-tree' -->
<li role="treeitem card-treeitem" class="jstree-node" data-bind="css: {'jstree-open':
↳(card.tiles().length > 0 && card.expanded()), 'jstree-closed' : (card.tiles().length >
↳0 && !card.expanded()), 'jstree-leaf': card.tiles().length === 0}, scrollTo: card.
↳scrollTo, container: '.resource-editor-tree'">
  <i class="jstree-icon jstree-ocl" role="presentation" data-bind="click: function()
↳{card.expanded(!card.expanded())}"></i>
  <a class="jstree-anchor" href="#" tabindex="-1" data-bind="css: {'filtered': card.
↳highlight(), 'jstree-clicked': card.selected, 'child-selected': card.isChildSelected()}
↳, click: function () { card.canAdd() ? card.selected(true) : card.tiles()[0].
↳selected(true) }, ">
    <i class="fa fa-file-o" role="presentation" data-bind="css: {'filtered': card.
↳highlight(), 'has-provisional-edits fa-file': card.doesChildHaveProvisionalEdits()}"></i>
    <i>
      <span style="padding-right: 5px;" data-bind="text: card.model.name"></span>
      <!-- ko if: card.canAdd() -->
        <i class="fa fa-plus-circle add-new-tile" role="presentation" data-bind="css: {
↳'jstree-clicked': card.selected}" data-toggle="tooltip" data-original-title="{% trans
↳"Add New" %}"></i>
        <!-- /ko -->
      </a>
      <ul class="jstree-children" aria-expanded="true">
        <div data-bind="sortable: {
          data: card.tiles,
          beforeMove: self.beforeMove,
          afterMove: card.reorderTiles
        }">
          <li role="treeitem" class="jstree-node" data-bind="css: {'jstree-open':
↳(cards.length > 0 && expanded), 'jstree-closed' : (cards.length > 0 && !expanded()),
↳'jstree-leaf': cards.length === 0}">
            <i class="jstree-icon jstree-ocl" role="presentation" data-bind="click:
↳function(){expanded(!expanded())}"></i>
            <a class="jstree-anchor" href="#" tabindex="-1" data-bind="click:
↳function () { self.form.selection($data) }, css: {'jstree-clicked': selected, 'child-
↳selected': isChildSelected(), 'filtered-leaf': card.highlight()}">
              <i class="fa fa-file" role="presentation" data-bind="css: {'has-
↳provisional-edits': doesChildHaveProvisionalEdits() || $data.hasprovisionaledits()}"></i>
              <i>
                <strong style="margin-right: 10px;">
                  <!-- ko if: card.widgets().length > 0 -->
                  <span data-bind="text: card.widgets()[0].label || card.model.name
↳"></span>:
```

(continues on next page)

(continued from previous page)

```

<div style="display: inline;" data-bind="component: {
    name: self.form.widgetLookup[card.widgets()[0].widget_id()].
↪name,
    params: {
        tile: $data,
        node: self.form.nodeLookup[card.widgets()[0].node_id()],
        config: self.form.widgetLookup[card.widgets()[0].widget_
↪id()].config,
        label: self.form.widgetLookup[card.widgets()[0].widget_
↪id()].label,
        value: $data.data[card.widgets()[0].node_id()],
        type: 'resource-editor',
        state: 'display_value'
    }
}"></div>
<!-- /ko -->
<!-- ko if: card.widgets().length === 0 -->
<span data-bind="text: card.model.name"></span>
<!-- /ko -->
</strong>
</a>
<!-- ko if: cards.length > 0 -->
<ul class="jstree-children" aria-expanded="true" data-bind="foreach: {
    data: cards,
    as: 'card'
}">
    <!-- ko component: {
        name: self.form.cardComponentLookup[card.model.component_id()].
↪componentname,
        params: {
            state: 'editor-tree',
            card: card,
            tile: null,
            loading: self.loading,
            form: self.form
        }
    } --> <!-- /ko -->
</ul>
<!-- /ko -->
</li>
</div>
<!-- /ko -->
</ul>
</li>
<!-- /ko -->

<!-- ko if: state === 'designer-tree' -->
<li role="treeitem card-treeitem" class="jstree-node" data-bind="css: {'jstree-open':
↪((card.cards().length > 0 || card.widgets().length > 0) && card.expanded()), 'jstree-
↪closed' : ((card.cards().length > 0 || card.widgets().length > 0) && !card.expanded()),
↪'jstree-leaf': card.cards().length === 0 && card.widgets().length === 0}, scrollTo:
↪card.scrollTo, container: '.designer-card-tree'">

```

(continues on next page)

(continued from previous page)

```

    <i class="jstree-icon jstree-ocl" role="presentation" data-bind="click: function()
    ↪{card.expanded(!card.expanded())}"></i>
    <a class="jstree-anchor" href="#" tabindex="-1" data-bind="css:{'filtered': card.
    ↪highlight(), 'jstree-clicked': card.selected, 'child-selected': card.isChildSelected()}
    ↪, click: function () { card.selected(true) },">
        <i class="fa fa-file-o" role="presentation"></i>
        <span style="padding-right: 5px;" data-bind="text: card.model.name"></span>
    </a>
    <!-- ko if: card.cards().length > 0 || card.widgets().length > 0 -->
    <ul class="jstree-children card-designer-tree" aria-expanded="true">
        <div data-bind="sortable: {
            data: card.widgets,
            as: 'widget',
            beforeMove: self.beforeMove,
            afterMove: function() { card.model.save() }
        }">
            <li role="treeitem" class="jstree-node jstree-leaf" data-bind="css: {
                'jstree-last': $index() === (card.widgets().length - 1) && card.
    ↪cards().length === 0
            }">
                <i class="jstree-icon jstree-ocl" role="presentation"></i>
                <a class="jstree-anchor" href="#" tabindex="-1" data-bind="click:
    ↪function() { widget.selected(true) }, css:{'jstree-clicked': widget.selected, 'hover':
    ↪widget.hovered}, event: { mouseover: function(){ widget.hovered(true) }, mouseout:
    ↪function(){ widget.hovered(null) } }">
                    <i data-bind="css: widget.datatype.iconclass" role="presentation"></
    ↪i>
                    <strong style="margin-right: 10px;" >
                        <span data-bind="text: !!(widget.label()) ? widget.label() :
    ↪widget.node.name"></span>
                    </strong>
                </a>
            </li>
        </div>
        <div data-bind="sortable: {
            data: card.cards,
            as: 'childCard',
            beforeMove: self.beforeMove,
            afterMove: function() {
                card.reorderCards();
            }
        }">
            <div data-bind="css: {
                'jstree-last': ($index() === (card.cards().length - 1))
            }">
                <!-- ko component: {
                    name: self.form.cardComponentLookup[childCard.model.component_
    ↪id()].componentname,
                    params: {
                        state: 'designer-tree',
                        card: childCard,
                        tile: null,

```

(continues on next page)

(continued from previous page)

```

        loading: self.loading,
        form: self.form
    }
} --> <!-- /ko -->
</div>
</div>
</ul>
<!-- /ko -->
</li>
<!-- /ko -->

<!-- ko if: state === 'permissions-tree' -->
<li role="treeitem card-treeitem" class="jstree-node" data-bind="css: {'jstree-open':
↳ ((card.cards().length > 0 || card.widgets().length > 0) && card.expanded()), 'jstree-
↳ closed' : ((card.cards().length > 0 || card.widgets().length > 0) && !card.expanded()),
↳ 'jstree-leaf': card.cards().length === 0 && card.widgets().length === 0}">
    <i class="jstree-icon jstree-ocl" role="presentation" data-bind="click: function()
↳ {card.expanded(!card.expanded())}"></i>
    <a class="jstree-anchor permissions-card" href="#" tabindex="-1" data-bind="css:{
↳ 'jstree-clicked': card.selected, 'child-selected': card.isChildSelected()}, click:
↳ function () { card.selected(true) },">
        <i class="fa fa-file-o" role="presentation"></i>
        <span style="padding-right: 5px;" data-bind="text: card.model.name, css:{
↳ 'filtered': card.highlight()}">
            </span>
            <span class="node-permissions">
                <!--ko if: card.perms -->
                <!-- ko foreach: card.perms() -->
                <i class="node-permission-icon" data-bind="css: $data.icon"></i>
                <!-- /ko -->
                <!-- /ko -->
            </span>
        </a>
        <!-- ko if: card.cards().length > 0 || card.widgets().length > 0 -->
        <ul class="jstree-children" aria-expanded="true">
            <div data-bind="sortable: {
                data: card.widgets,
                as: 'widget',
                beforeMove: self.beforeMove,
                afterMove: function() { card.model.save() }
            }">
                <li role="treeitem" class="jstree-node jstree-leaf" data-bind="css: {
                    'jstree-last': $index() === (card.widgets().length - 1) && card.
↳ cards().length === 0
                }">
                    <i class="jstree-icon jstree-ocl" role="presentation"></i>
                    <a class="jstree-anchor permissions-widget" href="#" tabindex="-1">
                        <i class="fa fa-file" role="presentation" ></i>
                        <strong style="margin-right: 10px;" >
                            <span data-bind="text: !(widget.label()) ? widget.label() :
↳ widget.node.name"></span>

```

(continues on next page)

(continued from previous page)

```

        </strong>
      </a>
    </li>
  </div>
  <div data-bind="foreach: {
    data: card.cards,
    as: 'card'
  }">
    <!-- ko component: {
      name: self.form.cardComponentLookup[card.model.component_id()].
↪componentname,
      params: {
        state: 'permissions-tree',
        card: card,
        tile: null,
        loading: self.loading,
        form: self.form,
        multiselect: true
      }
    } --> <!-- /ko -->
  </div>
</ul>
<!-- /ko -->
</li>
<!-- /ko -->

<!-- ko if: state === 'form' -->
<div class="card-component">

  <!--ko if: reviewer && provisionalTileViewModel.selectedProvisionalEdit() -->
  <div class="edit-message-container">
    <span>{% trans 'Currently showing edits by' %}</span>
    <span class="edit-message-container-user" data-bind="text:↪
↪provisionalTileViewModel.selectedProvisionalEdit().username() + '.'"></span>
    <!--ko if: !provisionalTileViewModel.tileIsFullyProvisional() -->
    <a class="reset-authoritative" href="#" data-bind="click: function()
↪{provisionalTileViewModel.resetAuthoritative();}">{% trans 'Return to approved edits'
↪%}</a>
    <!--/ko-->
    <!--ko if: provisionalTileViewModel.selectedProvisionalEdit().isfullyprovisional↪
↪-->
    <span>{% trans ' This is a new contribution by a provisional editor.' %}</span>
    <!--/ko-->
  </div>
  <!--/ko-->

  <!--ko if: reviewer && provisionalTileViewModel.provisionaledits().length > 0 && !
↪provisionalTileViewModel.selectedProvisionalEdit()-->
  <div class="edit-message-container approved">
    <div>{% trans 'Currently showing the most recent approved edits' %}</div>
  </div>

```

(continues on next page)

(continued from previous page)

```

<!--/ko-->

<div class="new-provisional-edit-card-container">
  <!--ko if: reviewer && provisionalTileViewModel.provisionaledits().length > 0 -->
  <!--ko if: !provisionalTileViewModel.tileIsFullyProvisional() -->
  <div class='new-provisional-edits-list'>
    <div class='new-provisional-edits-header'>
      <div class='new-provisional-edits-title'>{% trans 'Provisional Edits' %}
    </div>
    <div class="btn btn-shim btn-danger btn-labeled btn-xs fa fa-trash new-
    provisional-edits-delete-all" style="padding: 3px;" data-bind="click: function()
    {provisionalTileViewModel.deleteAllProvisionalEdits()}">{% trans 'Delete all edits' %}
    </div>
    </div>
    <!--ko foreach: { data: provisionalTileViewModel.provisionaledits(), as: 'pe' -->
    <div class='new-provisional-edit-entry' data-bind="css: {'selected': pe ===
    $parent.provisionalTileViewModel.selectedProvisionalEdit()}, click: function(){$parent.
    provisionalTileViewModel.selectProvisionalEdit(pe)}">
      <div class='title'>
        <div class='field'>
          <span data-bind="text : pe.username"></span>
        </div>
        <a href='' class='field fa fa-times-circle new-delete-provisional-
        edit' data-bind="click : function(){$parent.provisionalTileViewModel.
        rejectProvisionalEdit(pe)}"></a>
      </div>
      <div class="field timestamp">
        <span data-bind="text : pe.displaydate"></span>
        <span data-bind="text : pe.displaytimestamp"></span>
      </div>
    </div>
    <!-- /ko -->
  </div>
  <!--/ko-->
  <!--/ko-->

  <div class="card">

    <h4 data-bind="text: card.model.name"></h4>
    <h5 data-bind="text: card.model.instructions"></h5>

    <!-- ko if: card.widgets().length > 0 -->
    <form class="widgets" style="margin-bottom: 20px;">
      <div data-bind="foreach: {
        data:card.widgets, as: 'widget'
      }">
        <div data-bind='component: {
          name: self.form.widgetLookup[widget.widget_id()].name,

```

(continues on next page)

(continued from previous page)

```

        params: {
            formData: self.tile.formData,
            tile: self.tile,
            form: self.form,
            config: widget.configJSON,
            label: widget.label(),
            value: self.tile.data[widget.node_id()],
            node: self.form.nodeLookup[widget.node_id()],
            expanded: self.expanded,
            graph: self.form.graph,
            type: "resource-editor"
        }
    }, css:{ "active": widget.selected, "hover": widget.hovered, "widget-
    preview": self.preview
    }, click: function(data, e) { if (!widget.selected() && self.preview)
    {widget.selected(true);}
    }, event: { mouseover: function(){ if (self.preview){widget.hovered(true) } }
    , mouseout: function(){ if (self.preview){widget.hovered(null)} } }'</div>
    </div>
</form>
<!-- /ko -->
<!-- ko if: card.widgets().length === 0 -->
<ul class="card-summary-section" data-bind="css: {disabled: !tile.tileid}">
    <!-- ko foreach: { data: tile.cards, as: 'card' } -->
    <li class="card-summary">
        <a href="javascript:void(0)" data-bind="click: function () {
            if (card.parent.tileid) {
                card.canAdd() ? card.selected(true) : card.tiles()[0].
    selected(true);
            }
        }">
        <h4 class="card-summary-name">
            <span data-bind="text: card.model.name"></span>
            <!-- ko if: card.canAdd() && card.parent.tileid -->
            <i class="fa fa-plus-circle card-summary-add"></i>
            <!-- /ko -->
        </h4>
        </a>
        <ul class="tile-summary-item" data-bind="foreach: {
            data: card.tiles,
            as: 'tile'
        }">
            <li class="tile-summary">
                <a href="#" data-bind="click: function () { tile.
    selected(true) }">
                    <!-- ko if: card.widgets().length > 0 -->
                    <span data-bind="text: card.widgets()[0].label || card.
    model.name" class="tile-summary-label"></span>:
                    <div style="display: inline;" data-bind="component: {
                        name: self.form.widgetLookup[card.widgets()[0].
    widget_id()].name,
                        params: {

```

(continues on next page)

(continued from previous page)

```

        tile: tile,
        node: self.form.nodeLookup[card.widgets()[0]].

↪node_id()],
        config: self.form.widgetLookup[card.widgets()[0]].
↪widget_id()].config,
        label: self.form.widgetLookup[card.widgets()[0]].
↪widget_id()].label,
        value: tile.data[card.widgets()[0].node_id()],
        type: 'resource-editor',
        state: 'display_value'
    }
}"}</div>
<!-- /ko -->
<!-- ko if: card.widgets().length === 0 -->
<span data-bind="text: card.model.name"></span>
<!-- /ko -->
</a>
</li>
</ul>
</li>
<!-- /ko -->
</ul>
<!-- /ko -->
<div class="install-buttons">
    <!-- ko if: tile.tileid -->
    <button class="btn btn-shim btn-warning btn-labeled btn-lg fa fa-trash"
↪data-bind="click: function () { self.form.deleteTile(tile); }">{% trans 'Delete this
↪record' %}</button>
    <!-- /ko -->
    <!-- ko if: tile.dirty() -->
    <!-- ko if: provisionalTileViewModel && !provisionalTileViewModel.
↪tileIsFullyProvisional() -->
    <button class="btn btn-shim btn-danger btn-labeled btn-lg fa fa-times"
↪data-bind="click: tile.reset">{% trans 'Cancel edit' %}</button>
    <!-- /ko -->
    <!-- ko if: tile.tileid -->
    <button class="btn btn-shim btn-mint btn-labeled btn-lg fa fa-plus"
↪data-bind="click: function () { self.form.saveTile(tile); }">{% trans 'Save edit' %}</
↪button>
    <!-- /ko -->
    <!-- /ko -->
    <!-- ko if: !tile.tileid -->
    <button class="btn btn-shim btn-mint btn-labeled btn-lg fa fa-plus" data-
↪bind="click: function () { self.form.saveTile(tile); }">{% trans 'Add' %}</button>
    <!-- /ko -->
</div>

</div>
</div>
</div>
<!-- /ko -->

```

(continues on next page)

(continued from previous page)

```

<!-- ko if: state === 'report' -->
<div class="rp-card-section">
  <span class="rp-tile-title" data-bind="text: card.model.get('name')"></span>
  <!-- ko foreach: { data: card.tiles, as: 'tile' } -->
    <div class="rp-card-section">
      <!-- ko if: card.model.get('widgets')().length > 0 -->
        <div class="rp-report-tile" data-bind="attr: { id: tile.tileid }">
          <dl class="dl-horizontal">
            <!-- ko foreach: { data: card.model.get('widgets'), as: 'widget' } -
      ↪ ->
              <!-- ko component: {
                  name: widget.widgetLookup[widget.get("widget_id")()].
      ↪ name,
                  params: {
                      config: configJSON,
                      label: widget.get("label")(),
                      node: widget.node,
                      value: tile.data[widget.node.nodeid],
                      state: "report"
                  }
              } --><!-- /ko -->
            <!-- /ko -->
          </dl>
        </div>
      <!-- /ko -->

      <div class="rp-report-container-tile" data-bind="visible: card.cards().
      ↪ length > 0">
        <!-- ko foreach: { data: tile.cards, as: 'card' } -->
          <!-- ko component: {
              name: card.model.cardComponentLookup[card.model.component_
      ↪ id()].componentname,
              params: {
                  state: 'report',
                  card: card
              }
          } --> <!-- /ko -->
        <!-- /ko -->
      </div>
    </div>
  <!-- /ko -->

  <!-- ko if: card.tiles().length === 0 -->
  <div class="row rp-report-tile rp-no-data">
    <!-- ko ifnot: card.model.get('cardid') -->
      <% trans "Sorry, you don't have access to this information" %>
    <!-- /ko -->
    <!-- ko if: card.model.get('cardid') -->
      <% trans "No data added yet for" %> " <span data-bind="text: card.model.get('name
      ↪ ')"></span>"
    <!-- /ko -->
  </div>

```

(continues on next page)

(continued from previous page)

```

    <!-- /ko -->
</div>
<!-- /ko -->

<!-- /ko -->

```

And here's the Knockout file:

```

define([
    'knockout',
    'templates/views/components/cards/default.htm',
    'bindings/scrollTo'
], function(ko, defaultCardTemplate) {
    var viewModel = function(params) {
        this.state = params.state || 'form';
        this.preview = params.preview;
        this.loading = params.loading || ko.observable(false);
        this.card = params.card;
        this.tile = params.tile;
        if (this.preview) {
            if (!this.card.newTile) {
                this.card.newTile = this.card.getNewTile();
            }
            this.tile = this.card.newTile;
        }
        this.form = params.form;
        this.provisionalTileViewModel = params.provisionalTileViewModel;
        this.viewer = params.viewer;
        this.expanded = ko.observable(true);
        this.beforeMove = function(e) {
            e.cancelDrop = (e.sourceParent !== e.targetParent);
        };
    };
    return ko.components.register('default-card', {
        viewModel: viewModel,
        template: defaultCardTemplate,
    });
});

```

Registering your Card Component

To register your Component, you'll need a JSON configuration file looking a lot like this sample:

```

{
  "name": "My New Card",
  "componentid": "eea17d6c-0c32-4536-8a01-392df734de1c",
  "component": "/views/components/cards/my-new-card",
  "componentname": "my-new-card",
  "description": "An awesome new card that does wonderful things.",
  "defaultconfig": {}
}

```

componentid

Optional A UUID4 for your Component. Feel free to generate one in advance if that fits your workflow; if not, Arches will generate one for you and print it to STDOUT when you register the Component.

name

Required The name of your new Card Component, visible in the drop-down list of card components in the Arches Designer.

description

Required A brief description of your component.

component

Required The path to the component view you have developed. Example: `views/components/cards/sample-datatype`

componentname

Required Set this to the last part of `component` above.

defaultconfig

Required You can provide user-defined default configuration here. Make it a JSON dictionary of keys and values. An empty dictionary is acceptable.

Card Commands

To register your Card Component, use this command:

```
python manage.py card_component register --source /Documents/projects/mynewproject/
↪mynewproject/cards/new-card-component.json
```

The command will confirm your Component has been registered, and you can also see it with:

```
python manage.py card_component list
```

If you make an update to your Card Component, you can load the changes to Arches with:

```
python manage.py card_component update --source /Documents/projects/mynewproject/
↪mynewproject/cards/new-card-component.json
```

All the Card Component commands are detailed in [Command Line Reference - Card Component Commands](#).

Datatypes

A `DataType` defines a type of business data. `DataTypes` are associated with `Nodes` and `Widgets`. When you are designing your Cards, the `Widgets` with the same `DataType` as the `Node` you are collecting data for will be available. In your Branches, each `Node` with a `DataType` will honor the `DataType` configuration you specify when you create it.

The simplest (non-configurable, non-searchable) `DataTypes` consist of a single Python file. If you want to provide `Node`-specific configuration to your `DataType` (such as whether to expose a `Node` with that `DataType` to Advanced Search or how the data is rendered), you'll also develop a UI component comprising a Django template and JavaScript file.

In your Project, these files must be placed accordingly:

Optional Configuration Component:

```
/myproject/myproject/media/js/views/components/datatypes/sample_datatype.js
/myproject/myproject/templates/views/components/datatypes/sample_datatype.htm
```

DataType File:

```
/myproject/myproject/datatypes/sample_datatype.py
```

To begin, let's examine the sample-datatype included with Arches:

```
1 from arches.app.datatypes.base import BaseDataType
2 from arches.app.models import models
3 from arches.app.models.system_settings import settings
4
5 sample_widget = models.Widget.objects.get(name="sample-widget")
6
7 details = {
8     "datatype": "sample-datatype",
9     "iconclass": "fa fa-file-code-o",
10    "modulename": "datatypes.py",
11    "classname": "SampleDataType",
12    "defaultwidget": sample_widget,
13    "defaultconfig": {"placeholder_text": ""},
14    "configcomponent": "views/components/datatypes/sample-datatype",
15    "configname": "sample-datatype-config",
16    "isgeometric": False,
17    "issearchable": False,
18 }
19
20
21 class SampleDataType(BaseDataType):
22     def validate(self, value, row_number=None, source=None):
23         errors = []
24         try:
25             value.upper()
26         except:
27             errors.append(
28                 {
29                     "type": "ERROR",
30                     "message": "datatype: {0} value: {1} {2} {3} - {4}. {5}".format(
31                         self.datatype_model.datatype, value, row_number, source, "this_
↪is not a string", "This data was not imported.",
32                 ),
33             )
34         return errors
35
36     def append_to_document(self, document, nodevalue, nodeid, tile):
37         document["strings"].append({"string": nodevalue, "nodegroup_id": tile.nodegroup_
↪id})
38
39     def transform_export_values(self, value, *args, **kwargs):
40         if value != None:
41             return value.encode("utf8")
42
43     def get_search_terms(self, nodevalue, nodeid=None):
```

(continues on next page)

(continued from previous page)

```

45     terms = []
46     if nodevalue is not None:
47         if settings.WORDS_PER_SEARCH_TERM == None or (len(nodevalue.split(" ")) <
↳ settings.WORDS_PER_SEARCH_TERM):
48             terms.append(nodevalue)
49     return terms
50
51     def append_search_filters(self, value, node, query, request):
52         try:
53             if value["val"] != "":
54                 match_type = "phrase_prefix" if "~" in value["op"] else "phrase"
55                 match_query = Match(field="tiles.data.%s" % (str(node.pk)), query=value[
↳ "val"], type=match_type,)
56                 if "!" in value["op"]:
57                     query.must_not(match_query)
58                     query.filter(Exists(field="tiles.data.%s" % (str(node.pk))))
59                 else:
60                     query.must(match_query)
61         except KeyError:
62             pass

```

Writing Your DataType

Your DataType needs, at minimum, to implement the `validate` method. You're also likely to implement the `transform_import_values` or `transform_export_values` methods. Depending on whether your DataType is spatial, you may need to implement some other methods as well. If you want to expose Nodes of your DataType to Advanced Search, you'll also need to implement the `append_search_filters` method.

You can get a pretty good idea of what methods you need to implement by looking at the `BaseDataType` class in the Arches source code located at `arches/app/datatypes/base.py` and below:

```

1  import json
2  from django.core.urlresolvers import reverse
3  from arches.app.models import models
4
5  class BaseDataType(object):
6
7      def __init__(self, model=None):
8          self.datatype_model = model
9
10     def validate(self, value, row_number=None, source=None):
11         return []
12
13     def append_to_document(self, document, nodevalue, nodeid, tile):
14         """
15         Assigns a given node value to the corresponding key in a document in
16         in preparation to index the document
17         """
18         pass
19
20     def after_update_all(self):

```

(continues on next page)

(continued from previous page)

```

21     """
22     Refreshes mv_geojson_geoms materialized view after save.
23     """
24     pass
25
26     def transform_import_values(self, value, nodeid):
27         """
28         Transforms values from probably string/wkt representation to specified
29         datatype in arches
30         """
31         return value
32
33     def transform_export_values(self, value, *args, **kwargs):
34         """
35         Transforms values from probably string/wkt representation to specified
36         datatype in arches
37         """
38         return value
39
40     def get_bounds(self, tile, node):
41         """
42         Gets the bounds of a geometry if the datatype is spatial
43         """
44         return None
45
46     def get_layer_config(self, node=None):
47         """
48         Gets the layer config to generate a map layer (use if spatial)
49         """
50         return None
51
52     def should_cache(self, node=None):
53         """
54         Tells the system if the tileserver should cache for a given node
55         """
56         return False
57
58     def should_manage_cache(self, node=None):
59         """
60         Tells the system if the tileserver should clear cache on edits for a
61         given node
62         """
63         return False
64
65     def get_map_layer(self, node=None):
66         """
67         Gets the array of map layers to add to the map for a given node
68         should be a dictionary including (as in map_layers table):
69         nodeid, name, layerdefinitions, isoverlay, icon
70         """
71         return None
72

```

(continues on next page)

(continued from previous page)

```

73 def clean(self, tile, nodeid):
74     """
75     Converts " values to null when saving a tile.
76     """
77     if tile.data[nodeid] == '':
78         tile.data[nodeid] = None
79
80 def get_map_source(self, node=None, preview=False):
81     """
82     Gets the map source definition to add to the map for a given node
83     should be a dictionary including (as in map_sources table):
84     name, source (json)
85     """
86     tileserver_url = reverse('tileserver')
87     if node is None:
88         return None
89     source_config = {
90         "type": "vector",
91         "tiles": ["%s/%s/{z}/{x}/{y}.pbf" % (tileserver_url, node.nodeid)]
92     }
93     count = None
94     if preview == True:
95         count = models.TileModel.objects.filter(data__has_key=str(node.nodeid)).
↪count()
96     if count == 0:
97         source_config = {
98             "type": "geojson",
99             "data": {
100                 "type": "FeatureCollection",
101                 "features": [
102                     {
103                         "type": "Feature",
104                         "properties": {
105                             "total": 1
106                         },
107                         "geometry": {
108                             "type": "Point",
109                             "coordinates": [
110                                 -122.4810791015625,
111                                 37.93553306183642
112                             ]
113                         }
114                     },
115                     {
116                         "type": "Feature",
117                         "properties": {
118                             "total": 100
119                         },
120                         "geometry": {
121                             "type": "Point",
122                             "coordinates": [
123                                 -58.30078125,

```

(continues on next page)

(continued from previous page)

```

124         -18.075412438417395
125     ]
126 }
127 },
128 {
129     "type": "Feature",
130     "properties": {
131         "total": 1
132     },
133     "geometry": {
134         "type": "LineString",
135         "coordinates": [
136             [
137                 -179.82421875,
138                 44.213709909702054
139             ],
140             [
141                 -154.16015625,
142                 32.69486597787505
143             ],
144             [
145                 -171.5625,
146                 18.812717856407776
147             ],
148             [
149                 -145.72265625,
150                 2.986927393334876
151             ],
152             [
153                 -158.37890625,
154                 -30.145127183376115
155             ]
156         ]
157     }
158 },
159 {
160     "type": "Feature",
161     "properties": {
162         "total": 1
163     },
164     "geometry": {
165         "type": "Polygon",
166         "coordinates": [
167             [
168                 [
169                     -50.9765625,
170                     22.59372606392931
171                 ],
172                 [
173                     -23.37890625,
174                     22.59372606392931
175                 ],

```

(continues on next page)

(continued from previous page)

```

176         [
177             -23.37890625,
178             42.94033923363181
179         ],
180         [
181             -50.9765625,
182             42.94033923363181
183         ],
184         [
185             -50.9765625,
186             22.59372606392931
187         ]
188     ]
189 }
190 },
191 {
192     "type": "Feature",
193     "properties": {
194         "total": 1
195     },
196     "geometry": {
197         "type": "Polygon",
198         "coordinates": [
199             [
200                 [
201                     -27.59765625,
202                     -14.434680215297268
203                 ],
204                 [
205                     -24.43359375,
206                     -32.10118973232094
207                 ],
208                 [
209                     0.87890625,
210                     -31.653381399663985
211                 ],
212                 [
213                     2.28515625,
214                     -12.554563528593656
215                 ],
216                 [
217                     -14.23828125,
218                     -0.3515602939922709
219                 ],
220                 [
221                     -27.59765625,
222                     -14.434680215297268
223                 ]
224             ]
225         ]
226     }
227 }

```

(continues on next page)

(continued from previous page)

```

228         }
229     ]
230 }
231 }
232 return {
233     "nodeid": node.nodeid,
234     "name": "resources-%s" % node.nodeid,
235     "source": json.dumps(source_config),
236     "count": count
237 }
238
239 def get_pref_label(self, nodevalue):
240     """
241     Gets the prefLabel of a concept value
242     """
243     return None
244
245 def get_display_value(self, tile, node):
246     """
247     Returns a list of concept values for a given node
248     """
249     return unicode(tile.data[str(node.nodeid)])
250
251 def get_search_terms(self, nodevalue, nodeid=None):
252     """
253     Returns a nodevalue if it qualifies as a search term
254     """
255     return []
256
257 def append_search_filters(self, value, node, query, request):
258     """
259     Allows for modification of an elasticsearch bool query for use in
260     advanced search
261     """
262     pass
263
264 def handle_request(self, current_tile, request, node):
265     """
266     Updates files
267     """
268     pass

```

the validate method

Here, you write logic that the Tile model will use to accept or reject a Node's data before saving. This is the core implementation of what your DataType is and is not.

The `validate` method returns an array of errors. If the array is empty, the data is considered valid. You can populate the errors array with any number of dictionaries with a `type` key and a `message` key. The value for `type` will generally be `ERROR`, but you can provide other kinds of messages.

the append_search_filters method

In this method, you'll create an Elasticsearch query Nodes matching this datatype based on input from the user in the Advanced Search screen. (You design this input form in your DataType's front-end component.)

Arches has its own Elasticsearch query [DSL builder class](#). You'll want to review that code for an idea of what to do. The search view passes your DataType a `Bool()` query from this class, which you call directly. You can invoke its `must`, `filter`, `should`, or `must-not` methods and pass complex queries you build with the DSL builder's `Match` class or similar. You'll execute this search directly in your `append_search_filters` method.

In-depth documentation of this part is planned, but for now, look at the [core datatypes](#) located in Arches' source code for examples of the approaches you can take here.

Note: If you're an accomplished Django developer, it should also be possible to use Elastic's own [Python DSL builder](#) in your Project to build the complex search logic you'll pass to Arches' `Bool()` search, but this has not been tested.

Configuring your DataType

You'll need to populate the `details` dictionary to configure your new DataType.

```
details = {
    "datatype": "sample-datatype",
    "iconclass": "fa fa-file-code-o",
    "modulename": "datatypes.py",
    "classname": "SampleDataType",
    "defaultwidget": sample_widget,
    "defaultconfig": {"placeholder_text": ""},
    "configcomponent": "views/components/datatypes/sample-datatype",
    "configname": "sample-datatype-config",
    "isgeometric": False,
    "issearchable": False,
```

datatype

Required The name of your datatype. The convention in Arches is to use *kebab-case* here.

iconclass

Required The FontAwesome icon class your DataType should use. Browse them [here](#).

modulename

Required This should always be set to `datatypes.py` unless you've developed your own Python module to hold your many DataTypes, in which case you'll know what to put here.

classname

Required The name of the Python class implementing your datatype, located in your DataType's Python file below these details.

defaultwidget

Required The default Widget to be used for this DataType.

defaultconfig

Optional You can provide user-defined default configuration here.

configcomponent

Optional If you develop a configuration component, put the fully-qualified name of the view here.
Example: `views/components/datatypes/sample-datatype`

configname

Optional The name of the Knockout component you have registered in your UI component's JavaScript file.

isgeometric

Required Used by the Arches UI to determine whether to create a Map Layer based on the DataType, and also for caching. If you're developing such a DataType, set this to True.

issearchable

Optional Determines if the datatype participates in advanced search. The default is false.

Important: `configcomponent` and `configname` are required together.

Developing the Configuration Component

Your component JavaScript file should register a Knockout component with your DataType's `configname`. This component should be an object with two keys: `viewModel`, and `template`

The value for `viewModel` should be a function where you put the logic for your template. You'll be setting up Knockout observable and computed values tied to any form elements you've developed to collect Advanced Search or Node-level configuration information from the user.

The value for `template` should be another object with the key `require`, and the value should be `text!datatype-config-templates/<your-datatype-name>`. Arches will know what to do with this – it comes from the value you supplied in your Python file's details dictionary for `configcomponent`.

Pulling it all together, here's the JavaScript portion of Arches' `date` DataType.

```
define(['knockout', 'datatype-config-templates/date.htm'], function (ko, dateTemplate) {
  var name = 'date-datatype-config';
  ko.components.register(name, {
    viewModel: function(params) {
      var self = this;
      this.search = params.search;
      if (this.search) {
        var filter = params.filterValue();
        this.viewMode = 'days';
        this.op = ko.observable(filter.op || '');
        this.searchValue = ko.observable(filter.val || '');
        this.filterValue = ko.computed(function () {
          return {

```

(continues on next page)

(continued from previous page)

```

        op: self.op(),
        val: self.searchValue()
    }
    }).extend({ throttle: 750 });
    params.filterValue(this.filterValue());
    this.filterValue.subscribe(function (val) {
        params.filterValue(val);
    });
}
},
template: dateTemplate,
});
return name;
});

```

Advanced Search Rendering

If you're supporting Advanced Search functionality for Nodes with your DataType, your Django template will include a search block, conditionally rendered by Knockout.js if the search view is active. Here's the one from the boolean datatype:

```

<!-- ko if: $data.search -->
{% block search %}
<div class="col-sm-12">
    <select class="resources" data-bind="value: searchValue, chosen: {width: '100%',
    ↪disable_search_threshold: 15}, options: [{id:'t', name:trueLabel}, {id:'f',
    ↪name:falseLabel}], optionsText: 'name', optionsValue: 'id'">

    </select>
</div>
{% endblock search %}
<!-- /ko -->

```

Note the `<!-- ko if: $data.search -->` directive opening and closing the search block. This is not an HTML comment – it's Knockout.js-flavored markup for the conditional rendering.

Arches' built-in date DataType does not use the Django template block directive, but only implements advanced search, and contains a more sophisticated example of the component logic needed:

```

{% load i18n %}
<!-- ko if: $data.search -->
<div class="col-md-4 col-lg-3">
    <select class="resources" tabindex="-1" style="display: none;" data-bind="value: op,
    ↪chosen: {width: '100%', disable_search_threshold: 15}">
        <option value="eq"> = </option>
        <option value="gt"> > </option>
        <option value="lt"> < </option>
        <option value="gte"> >= </option>
        <option value="lte"> <= </option>
    </select>
</div>

```

(continues on next page)

(continued from previous page)

```
<div class="col-md-8 col-lg-9">
  <input type="" placeholder="{% trans "Date" %}" class="form-control input-md widget-
↳input" data-bind="value: searchValue, datepicker: {format: 'YYYY-MM-DD', viewMode:
↳viewMode, minDate: false, maxDate: false}">
</div>
<!-- /ko -->
```

Node-specific Configuration

This section of your template should be enclosed in Knockout-flavored markup something like: `<!-- ko if: $data.graph -->`, and in your Knockout function you should follow the convention and end up with something like `if (this.graph) {`

Here, you put form elements corresponding to any configuration you've implemented in your `DataType`. These should correspond to keys in your `DataType`'s `defaultconfig`.

Arches' boolean `DataType` has the following `defaultconfig`:

```
{'falseLabel': 'No', 'trueLabel': 'Yes'}
```

You can see the corresponding data bindings in the Django template:

```
<!-- ko if: $data.graph -->
<div class="control-label">
  {% trans "Label 'True'" %}
</div>
<div class="col-xs-12 pad-no crud-widget-container">
  <input type="" id="" class="form-control input-md widget-input" data-bind="value:
↳trueLabel, valueUpdate: 'keyup'">
</div>
<div class="control-label">
  {% trans "Label 'False'" %}
</div>
<div class="col-xs-12 pad-no crud-widget-container">
  <input type="" id="" class="form-control input-md widget-input" data-bind="value:
↳falseLabel, valueUpdate: 'keyup'">
</div>
<!-- /ko -->
```

And finally, here is the boolean `DataType`'s JavaScript file in its entirety:

```
define(['knockout', 'datatype-config-templates/boolean.htm'], function (ko,
↳booleanTemplate) {
  var name = 'boolean-datatype-config';
  ko.components.register(name, {
    viewModel: function(params) {
      var self = this;
      this.search = params.search;
      this.graph = params.graph;

      this.trueLabel = params.config ? params.config.trueLabel : params.node.
```

(continues on next page)

(continued from previous page)

```
↪config.trueLabel;
    this.falseLabel = params.config ? params.config.falseLabel : params.node.
↪config.falseLabel;

    if (this.search) {
        var filter = params.filterValue();
        this.searchValue = ko.observable(filter.val || '');
        this.filterValue = ko.computed(function () {
            return {
                val: self.searchValue()
            }
        });
        params.filterValue(this.filterValue());
        this.filterValue.subscribe(function (val) {
            params.filterValue(val);
        });
    }
},
template: booleanTemplate,
});
return name;
});
```

Registering your DataType

These commands are identical to working with Widgets, but you use the word `datatype` instead. Please refer to [Command Line Reference - Widget Commands](#).

ETL Modules

The ETL Modules allow a developer to define ETL ([extract, transform, load](#)) processes that fit the user's business case. Arches includes basic ETL modules. The modules can be accessed in the [Bulk Data Manager](#), which currently supports import, export, and edit. A user can add a custom module, in addition to the modules included in the Arches.

Creating an ETL Module

A module comprises three separate files, which should be seen as front-end/back-end complements. On the front-end, you will need a component made from a Django HTML template and JavaScript pair, which should share the same basename.

In your Project, these files must be placed accordingly:

```
/my_project/my_project/media/js/views/components/etl_modules/sample-etl-module.
js                               /my_project/my_project/templates/views/components/etl_modules/
sample-etl-module.htm
```

The third file is a Python file which contains a dictionary telling Arches some important details about your module, as well as its main logic.

```
/my_project/my_project/etl_modules/sample_etl_module.py
```

Defining the Details

The first step in creating a ETL Module is defining the details in the top of your Function's .py file. The details is also used to register you etl module during the package loading or *on the command line*.

```
details = {
    "etlmoduleid": "",
    "name": "Sample ETL Module",
    "description": "This module is a sample module",
    "etl_type": "import",
    "component": "views/components/etl_modules/sample-etl-module",
    "componentname": "sample-etl-module",
    "modulename": "sample_etl_module.py",
    "classname": "SampleEtlModule",
    "config": {"bgColor": "#f5c60a", "circleColor": "#f9dd6c"},
    "icon": "fa fa-upload",
    "slug": "sample-etl-module",
    "helpsortorder": 9,
    "helptemplate": "sample-etl-module-help"
}
```

etlmoduleid

Optional A UUID4 for your ETL Module. Feel free to generate one in advance if that fits your workflow; if not, Arches will generate one for you.

name

Required The name of your new ETL Module, visible in the icons in the Bulk Data Manager menu.

description

Required The description of your new ETL Module, visible in the icons in the Bulk Data Manager menu.

etl_type

Required The type of your new ETL Module, currently import, export, and edit are supported

component

Required The path to the component view you have developed. Example: views/components/etl_modules/sample-etl-module

componentname

Required Set this to the last part of component above.

classname

Required The name of the Python class implementing your ETL Module, located in your module's Python file below the details.

modulename

Required The name of the Python file implementing your ETL Module.

config

Required You can provide user-defined default configuration here. Make it a JSON dictionary of keys and values. An empty dictionary is acceptable.

icon

Required The icon visible in the icon in the Bulk Data Manager menu.

slug

Required The string that will be used in the url to access your ETL Module

helptemplate

Optional The help template for your etl module in the Arches help section

helpsortorder

Optional The order in which the ETL Module helps will be listed in the Arches help section

The config field

Though not required, typically the `config` will include `bgColor` and `circleColor` that will determine the background and the icon colors visible in the Bulk Data Manager.

The additional properties can be added, if you would like to set the default values or add your user-defined configuration. For example, the string editors have the field `updateLimit` (set to 5,000 by default) which will limit the number of edits in a single etl process.

Writing your ETL Module

In your module's Python code, you have access to all your server-side models.

The importers and editors follow the pattern of

- creating the intermediary data in `load_staging` table as the tile-like json format
- processing the data either before or after staging the data
- validating the data if necessary (and recording the errors in the `load_errors` table)
- saving the data in the `tile` table if there are no validation errors
- indexing the database
- The progress needs to be saved in `load_event` table, if you want to access the status and the information about the etl.

If you want to take advantage of the pattern, you can start your development by extending the `BaseImportModule` for an importer or `BaseBulkEditor` for an editor, which will provide the basic functionality such as `reverse` (undo the import or edit). Then, you may want to write your own functions or overwrite the existing ones such as `validate`, `read`, `preview`, or `write`, as well as `run_load_task_async` and `run_load_task` if you would like to utilize the celery task manager.

see the examples in the existing etl module such as `base_data_editor.py`

```
class BulkStringEditor(BaseBulkEditor):
    def validate(self, request):
        ...

    def validate_inputs(self, request):
        ...

    def edit_staged_data(self, cursor, graph_id, node_id, operation, language_code,
↳ pattern, new_text):
        ...

    def get_preview_data(self, node_id, search_url, language_code, operation, old_text,
↳ case_insensitive, whole_word):
        ...
```

(continues on next page)

(continued from previous page)

```

def preview(self, request):
    ...

def write(self, request):
    ...

@load_data_async
def run_load_task_async(self, request):
    ...

def run_load_task(self, userid, loadid, module_id, graph_id, node_id, operation,
↪ language_code, pattern, new_text, resourceids):
    ...

```

Also, you can find the related models in `models.py` (`LoadStaging`, `LoadErrors`, and `LoadEvent`).

Registering your ETL Module

To register your ETL Module, use this command:

```
python manage.py etl_module register --source /projects/my_project/my_project/etl_
↪ modules/sample_etl_module.py
```

The command will confirm your ETL Module has been registered, and you can also list the existing modules with:

```
python manage.py etl_module list
```

To unregister your ETL Module, you can load the changes to Arches with:

```
python manage.py etl_module unregister --name Sample ETL Module
```

Examples to Get Started with ETL Modules

As is the case with other custom components in Arches, an html file and a javascript file are needed to design the user interface of your custom component. To help guide development of a custom ETL module, you can look at the files associated with the **Tile Excel Loader** that comes standard with core Arches. These are the component files for that module:

- `tile-excel-importer.js`
- `tile-excel-importer.htm`

Note that the `tile-excel-importer.js` javascript file imports a view model called `excel-file-import.js` where most of the logic is located.

You will notice that there are calls to submit that send strings such as “read” and “write” back to the Arches server. These strings are passed back to your module’s python file. In other words, calling `await self.submit('start');` will call the corresponding start method in your module.

That flexibility gives you gives one a great deal of freedom to implement custom logic in an ETL module.

Functions

Functions are the most powerful extension to Arches. Functions associated with a Resource are called during various CRUD operations, and have access to any server-side model. Proficient Python/Django developers will find few limitations extending an Arches Project with Functions.

Function must be created, registered, and then associated with a Resource Model.

Creating a Function

A Function comprises three separate files, which should be seen as front-end/back-end complements. On the front-end, you will need a component made from a Django HTML template and JavaScript pair, which should share the same basename.

In your Project, these files must be placed like so:

```
/myproject/myproject/media/js/views/components/functions/spatial_join.js      /
myproject/myproject/templates/views/components/functions/spatial_join.htm
```

The third file is a Python file which contains a dictionary telling Arches some important details about your Function, as well as its main logic.

```
/myproject/myproject/functions/spatial_join.py
```

Note: As in the example above, its advisable that all of your files share the same basename. (If your Function is moved into a Package, this is necessary.) A new Project should have an example function in it whose files you can copy to begin this process.

Defining the Function's Details

The first step in creating a function is defining the details that are in the top of your Function's .py file.

```
details = {
    'name': 'Sample Function',
    'type': 'node',
    'description': 'Just a sample demonstrating node group selection',
    'defaultconfig': {"selected_nodegroup":""},
    'classname': 'SampleFunction',
    'component': 'views/components/functions/sample-function'
}
```

name

Required Name is used to unregister a function, and shows up in the `fn list` command.

type

Required As of version 4.2, this should always be set to `node`

description

Optional Add a description of what your Function does.

defaultconfig

Required A JSON object with any configuration needed to serve your function's logic

classname

Required The name of the python class that holds this Function's logic.

component

Required Canonical path to html/js component.

More about the defaultconfig field

Any configuration information you need your Function to access can be stored here. If your function needs to calculate something based on the value of an existing Node, you can refer to it here. Or, if you want your Function to e-mail an administrator whenever a specific node is changed, both the Node ID and the email address to be used are good candidates for storage in the defaultconfig dictionary.

The defaultconfig field serves both as a default, and as your user-defined schema for your function's configuration data. Your front-end component for the function will likely collect some of this configuration data from the user and store it in the config attribute of the pertinent FunctionXGraph.

Writing your Function Logic

In your Function's Python code, you have access to all your server-side models. You're basically able to extend Arches in any way you please. You may want to review the [Data Model](#) documentation.

Function Hooks

Your function needs to extend the BaseFunction class. Depending on what you are trying to do, you will need to implement the get, save, delete, on_import, and/or after_function_save methods.

```
class MyFunction(BaseFunction):

    def get(self):
        raise NotImplementedError

    def save(self, tile, request):
        raise NotImplementedError

    def delete(self, tile, request):
        raise NotImplementedError

    def on_import(self, tile):
        raise NotImplementedError

    def after_function_save(self, functionxgraph, request):
        raise NotImplementedError
```

Note: Not all of these methods are called in the current Arches software. You can also leave any of them unimplemented, and the BaseFunction class will raise a NotImplementedError for you. Arches is designed to gracefully ignore these exceptions for functions.

A detailed description of current functionality is below.

save and delete

The Tile object will look up all its Graph's associated Functions upon being saved. Before writing to the database, it calls each function's `save` method, passing itself along with the Django Request object. This is likely where the bulk of your function's logic will reside.

The Tile object similarly calls each of its graph's functions' `delete` methods with the same parameters. Here, you can execute any cleanup or other desired side effects of a Tile's deletion. Your `delete` implementation will have the same signature as `save`.

after_function_save

The Graph view passes a `FunctionXGraph` object to `after_function_save`, along with the request.

The `FunctionXGraph` object has a `config` attribute which stores that instance's version of the `defaultconfig` dictionary. This is a good opportunity, for example, to programmatically manipulate the Function's configuration based on the Graph or any other server-side object.

You can also write any general logic that you'd like to fire upon the assignment of a Function to a Resource.

on_import

The import module calls `on_import` if the file format is a JSON-formatted Arches file, and passes an associated Tile object.

CSV imports do not call this hook.

The UI Component

Having implemented your function's logic, it's time to develop the front-end components required to associate it with Resources and provide any configuration data.

The component you develop here will be rendered in the Resource Manager when you associate the function with a Resource, and this is where you'll put any forms or other UI artifacts used to configure the Function.

Developing your Function's UI component is very similar to developing *Widgets*. More specific guidelines are in progress, but for now, refer to the sample code in your project's `templates/views/components/functions/` directory, and gain a little more insight from the `templates/views/components/widgets/` directory. The complementary JavaScript examples will be located in `media/js/views/components/functions/` and `media/js/views/components/widgets` directories.

Registering Functions

First, list the names of functions you already have registered:

```
(ENV)$ python manage.py fn list
```

Now you can register your new function with

```
(ENV)$ python manage.py fn register --source <path to your function's .py file>
```

For example:


```
(ENV)$ python manage.py fn register --source /Documents/projects/mynewproject/
↪mynewproject/functions/sample_function.py
```

Now navigate to the Function Manager in the Arches Designer to confirm that your new function is there and functional. If it's not, you may want to unregister your function, make additional changes, and re-register it. To unregister your function, simply run

```
(ENV)$ python manage.py fn unregister --name 'Sample Function'
```

All commands are listed in *Command Line Reference - Function Commands*.

Plugins

Plugins allow a developer to create an independent page in Arches that is accessible from the main navigation menu. For example, you may need a customized way of visualizing your resource data. A plugin would enable you to design such an interface. Plugins, like widgets and card components rely only on front-end code. Ajax queries, generally calls to the API, must be used to access any server side data.

Registering your Plugin

To register your Plugin, you'll need a JSON configuration file looking a lot like this sample:

```
{
  "pluginid": "b122ede7-24a6-4fc5-a3cc-f95bfa28b1cf",
  "name": "Sample Plugin",
  "icon": "fa fa-share-alt",
  "component": "views/components/plugins/sample-plugin",
  "componentname": "sample-plugin",
  "config": {},
  "slug": "sample-plugin",
  "sortorder": 0
}
```

pluginid

Optional A UUID4 for your Plugin. Feel free to generate one in advance if that fits your workflow; if not, Arches will generate one for you and print it to STDOUT when you register the Plugin.

name

Required The name of your new Plugin, visible when a user hovers over the main navigation menu

icon

Required The icon visible in the main navigation menu.

component

Required The path to the component view you have developed. Example: views/components/plugins/sample-plugin

componentname

Required Set this to the last part of component above.

config

Required You can provide user-defined default configuration here. Make it a JSON dictionary of keys and values. An empty dictionary is acceptable.

slug

Required The string that will be used in the url to access your plugin

sortorder

Required The order in which your plugin will be listed if there are multiple plugins

Plugin Commands

To register your Plugin, use this command:

```
python manage.py plugin register --source /Documents/projects/mynewproject/mynewproject/
↳plugins/sample-plugin.json
```

The command will confirm your Plugin has been registered, and you can also see it with:

```
python manage.py plugin list
```

If you make an update to your Plugin, you can load the changes to Arches with:

```
python manage.py plugin update --source /Documents/projects/mynewproject/mynewproject/
↳plugins/sample-plugin.json
```

Resource Reports

Arches enables projects to have custom reports on a per-resource model basis. Below is a guide to create and implement a custom resource report.

In your project, you'll need to create files in the following directories. If any directories listed here do not exist in your project, create them first.

```
my_proj/my_proj/reports/custom_report.json
my_proj/my_proj/media/js/reports/custom_report.js
my_proj/my_proj/templates/views/report-templates/custom_report.htm
```

Sample report .json file:

```
{
  "name": "My Custom Report Name",
  "componentid": "aee56c3a-44cf-4ab2-a5fb-6c51cda7b760",
  "component": "reports/custom_report",
  "componentname": "custom_report",
  "description": "A custom report.",
  "defaultconfig": {}
}
```

Sample report .js file:

```
define([
  'knockout',
  'viewmodels/report',
  'templates/views/report-templates/custom_report.htm'
], function(ko, ReportViewModel, customReportTemplate) {
  return ko.components.register('custom_report', {
```

(continues on next page)

(continued from previous page)

```
viewModel: function(params) {
  params.configKeys = [];
  var self = this;
  // define params for custom report here

  ReportViewModel.apply(this, [params]);
  // Put custom report logic here
},
template: customReportTemplate,
});
});
```

Sample report .htm file (note that extending the core arches default report is optional. See core arches default report for reference on overriding specific tagged sections, e.g. “{% block header %}”).:

```
{% extends "views/report-templates/default.htm" %}
{% load i18n %}

{% block body %}
  <!--ko if: hasProvisionalData() && (editorContext === false) -->
  <div class="report-provisional-flag">{% trans 'This resource has provisional edits_
↳(not displayed in this report) that are pending review' %}</div>
  <!--/ko-->
  <!--ko if: hasProvisionalData() && (editorContext === true && report.userisreviewer_
↳=== true) -->
  <div class="report-provisional-flag">{% trans 'This resource has provisional edits_
↳(not displayed in this report) that are pending review' %}</div>
  <!--/ko-->
  <!--ko if: hasProvisionalData() && (editorContext === true && report.userisreviewer_
↳=== false) -->
  <div class="report-provisional-flag">{% trans 'This resource has provisional edits_
↳that are pending review' %}</div>
  <!--/ko-->

  <div class="rp-report-section relative rp-report-section-root">
    <div class="rp-report-section-title">
      <!-- ko foreach: { data: report.cards, as: 'card' } -->
      <!-- ko if: !(ko.unwrap(card.tiles).length > 0) -->
      <!-- ko if: $index() != 0 --><hr class="rp-tile-separator"><!-- /ko -->
      <div class="rp-card-section">
        <!-- ko component: {
          name: card.model.cardComponentLookup[card.model.component_id()].
↳componentname,
          params: {
            state: 'report',
            preview: $parent.report.preview,
            card: card,
            pageVm: $root,
            hideEmptyNodes: $parent.hideEmptyNodes
          }
        } --> <!-- /ko -->
      </div>
    </div>
  </div>
```

(continues on next page)

(continued from previous page)

```
        <!-- /ko -->
    <!-- /ko -->
</div>
</div>
{% endblock body %}
```

Before registering your report, ensure that named references to the various report files are consistent. For ease, it is recommended to use one single name for all files to match the component name. Check the named references in your .js file to your component as well as the template name in case you encounter issues later.

Registering your report:

```
(ENV) $ python manage.py report register -s ./my_proj/reports/custom_report.json
```

Finally, in the Arches Graph Designer interface, navigate to the “Cards” tab of the resource model this report is for, click the root/top node in the card tree (is the name of the graph/resource model) in the left-hand side. On the far-right you will see a heading “Report Configuration”. Select your custom report from the dropdown labeled “Template”, and save changes.

Troubleshooting Tips

- Ensure that all references to a component name are consistent.
- Ensure that references to a template (.htm file) are consistent.
- Ensure your report exists in your database by checking the “report_templates” table.

Further Interest

Because templates often call other templates, e.g. the default report template for a resource instance in turn calls the default card component template, it may be of interest to either override or create a custom component for cards which get rendered within resource reports.

Search Filters

<https://github.com/archesproject/arches-docs/issues/222>

Widgets

Widgets allow you to customize how data of a certain DataType is entered into Arches, and further customize how that data is presented in Reports. You might have several Widgets for a given DataType, depending on how you want the Report to look or to match the context of a certain Resource.

Widgets are primarily a UI artifact, though they are closely tied to their underlying DataType.

To develop a custom Widget, you’ll need to write three separate files, and place them in the appropriate directories. For the appearance and behavior of the Widget, you’ll need a component made of a Django template and JavaScript file placed like so:

```
project_name/templates/views/components/widgets/sample-widget.htm project_name/
media/js/views/components/widgets/sample-widget.js
```

To register and configure the Widget, you’ll need a JSON configuration file:

```
project_name/widgets/sample-widget.json
```

Configuring your Widget

To start, here is a sample Widget JSON file:

```
{
  "name": "sample-widget",
  "component": "views/components/widgets/sample-widget",
  "defaultconfig": {
    "x_placeholder": "Longitude",
    "y_placeholder": "Latitude"
  },
  "helptext": null,
  "datatype": "sample-datatype"
}
```

The most important field here is the `datatype` field. This controls where your Widget will appear in the Arches Resource Designer. Nodes each have a `DataType`, and Widgets matching that `DataType` will be available when you're designing your Cards. The value must match an existing `DataType` within Arches.

You can also populate the `defaultconfig` field with any configuration data you wish, to be used in your Widget's front-end component.

Designing Your Widget

Your Widget's template needs to include three Django template "blocks" for rendering the Widget in different contexts within Arches. These blocks are called **form**, **config_form**, and **report**. As you might guess from their names, **form** is rendered when your Widget appears on a Card for business data entry, **config_form** is rendered when you configure the Widget on a card when designing a Resource, and **report** controls how data from your Widget is presented in a Report.

Here is an example:

```
{% extends "views/components/widgets/base.htm" %}
{% load i18n %}

{% block form %}
<div class="row widget-wrapper">
  <div class="form-group">
    <label class="control-label widget-input-label" for="" data-bind="text:label"></
    <label>
    <div class="col-xs-12">
      <input type="number" data-bind="textInput: x_value, attr: {placeholder: x_
      <placeholder}" class="form-control input-lg widget-input" style="margin-bottom: 5px">
    </div>
    <div class="col-xs-12">
      <input type="number" data-bind="textInput: y_value, attr: {placeholder: y_
      <placeholder}" class="form-control input-lg widget-input" style="margin-bottom: 5px">
    </div>
    <div class="col-xs-12">
      <input type="text" data-bind="textInput: srid" class="form-control input-lg
      <widget-input" style="margin-bottom: 5px">
    </div>
    <div class="col-xs-12">
```

(continues on next page)

(continued from previous page)

```

        <label class="control-label widget-input-label" for="">Preview</label>
        <input disabled type="text" data-bind="textInput: preview" class="form-
↪control input-lg widget-input">
    </div>
</div>
</div>
{% endblock form %}

{% block config_form %}
<div class="control-label">
    {% trans "X Coordinate Placeholder" %}
</div>
<div class="col-xs-12 crud-widget-container">
    <input type="" placeholder="{% trans "Placeholder" %}" id="" class="form-control
↪input-md widget-input" data-bind="textInput: x_placeholder">
</div>
<div class="control-label">
    {% trans "Y Coordinate Placeholder" %}
</div>
<div class="col-xs-12 crud-widget-container">
    <input type="" placeholder="{% trans "Placeholder" %}" id="" class="form-control
↪input-md widget-input" data-bind="textInput: y_placeholder">
</div>
{% endblock config_form %}

{% block report %}
<dt data-bind="text: label"></dt>
<dd>
    <div style='margin-bottom:2px' data-bind="text: value"></div>
</dd>
{% endblock report %}

```

To pull it all together, you'll need to write a complementary JavaScript file. The Arches UI uses Knockout.js, and the best way to develop your Widget in a compatible way is to write a Knockout component with a `viewModel` corresponding to your Widget's view (the Django template).

Here is an example, continuing with our `sample-widget`:

```

define([
    'knockout',
    'underscore',
    'viewmodels/widget',
    'templates/views/components/widgets/sample-widget.htm'
], function (ko, _, WidgetViewModel, sampleWidgetTemplate) {
    /**
     * registers a text-widget component for use in forms
     * @function external:"ko.components".text-widget
     * @param {object} params
     * @param {string} params.value - the value being managed
     * @param {function} params.config - observable containing config object
     * @param {string} params.config().label - label to use alongside the text input
     * @param {string} params.config().placeholder - default text to show in the text
↪input

```

(continues on next page)

(continued from previous page)

```

*/
return ko.components.register('sample-widget', {
  viewModel: function(params) {
    params.configKeys = ['x_placeholder', 'y_placeholder'];
    WidgetViewModel.apply(this, [params]);
    var self = this;
    if (this.value()) {
      var coords = this.value().split('POINT(')[1].replace(')', '').split(' ');
      var srid = this.value().split(';')[0].split('=')[1]
      this.x_value = ko.observable(coords[0]);
      this.y_value = ko.observable(coords[1]);
      this.srid = ko.observable('4326');
    } else {
      this.x_value = ko.observable();
      this.y_value = ko.observable();
      this.srid = ko.observable('4326');
    };

    this.preview = ko.pureComputed(function() {
      var res = "SRID=" + this.srid() + ";POINT(" + this.x_value() + " " +
↪this.y_value() + ")"
      this.value(res);
      return res;
    }, this);
  },
  template: sampleWidgetTemplate,
});
});

```

Registering your Widget

After placing your Django template and JavaScript files in their respective directories, you are now ready to register your Widget:

```
python manage.py widget register --source /Documents/projects/mynewproject/mynewproject/
↪widgets/sample-widget.json
```

The command will confirm your Widget has been registered, and you can also see it with:

```
python manage.py widget list
```

If you make an update to your Widget, you can load the changes to Arches with:

```
python manage.py widget update --source /Documents/projects/mynewproject/mynewproject/
↪widgets/sample-widget.json
```

All the Widget commands are detailed in [Command Line Reference - Widget Commands](#).

Workflows

Workflows are a type of *Plugin* that can simplify the data entry process. A workflow is composed of one or more cards from a resource model, placing them in a step-through set of forms. This provides users the ability to create new resource instances without having to traverse card-by-card through the resource model tree.

In other words, instead of using this interface to create a new resource:

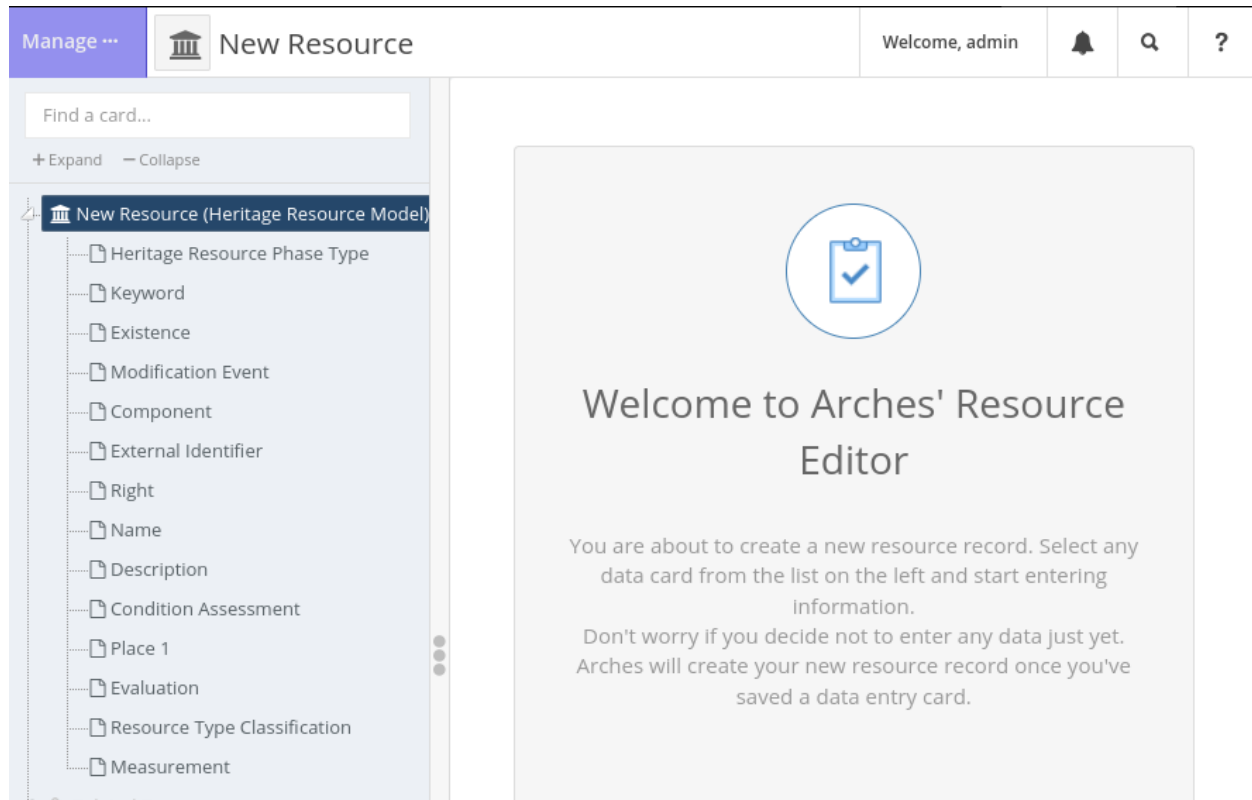


Fig. 36: Default full resource editor.

... a workflow can pare down the data entry interface to look something like this:

Workflows can be complex too, facilitating the creation of many different inter-related resource instances simultaneously. We'll use a very simple example here, however, to show how a workflow can be used to extract just a few cards from a large resource model to facilitate a "quick create" task that is easy for users to complete.

Creating a Workflow - the Basics

A very simple workflow will be presented here, based on the `arches-example-pkg` resource model called "Heritage Resource Model". This resource model has many cards, but we will make a workflow that pulls out just three of these cards—**Name/Name Type**, **Resource Type Classification**, and **Keyword**.

Workflows follow the standard extension pattern: an HTML/JS component and a JSON config. For this example, we have registration configs stored here:

```
my_project/plugins/quick-resource-create-workflow.json
```

and the main UI component looks like this:

Quick Create Resource

Welcome, admin

Quick Create Resource Workflow

Delete Workflow

Create Historic Resource

Add Keywords

Finish

Create historic resource here

Begin by providing the name and type of historic resource you are adding to the database.

dismiss

Name

Enter text

Name Type

Select an option

Fig. 37: A simple workflow abstracts data entry away from the card tree into forms.

```
my_project/templates/views/components/plugins/quick-resource-create-workflow.htm
my_project/media/js/views/components/plugins/quick-resource-create-workflow.js
```

Note: Remember, Workflows are just a special subset of Plugins, so the two types of extensions will be stored alongside each other.

In more advanced workflows, each step will have its own custom component, and these can be stored here:

```
my_project/templates/views/components/workflows/first-workflow-step.htm
my_project/media/js/views/components/workflows/first-workflow-step.js
```

These custom step components are discussed more [below](#), but our example won't use one.

Registration JSON

Because Workflows are just Plugins, their registration configurations are constructed the same. See [Registering your Plugin](#) for more about how to create the JSON file. For our purposes, it will look like this:

```
{
  "pluginid": "59b9a9cb-1654-43a6-abb7-7e4ca4c5bfea",
  "name": "Quick Create Resource",
  "icon": "fa fa-check",
  "component": "views/components/plugins/quick-resource-create-workflow",
  "componentname": "quick-resource-create-workflow",
  "config": {
```

(continues on next page)

(continued from previous page)

```
    "show": true
  },
  "slug": "quick-resource-create-workflow",
  "sortorder": 0
}
```

Main UI Component

The HTML for this component `quick-resource-create-workflow.htm` can be exceptionally simple (just two lines):

```
{% extends "views/components/plugins/workflow.htm" %}
{% load i18n %}
```

The workflow's behavior is defined in `quick-resource-create-workflow.js`. You'll begin with the boilerplate content below. Note that:

- The file name, registered component name, and `this.componentName` must all match.
- The `stepConfig` attribute will hold the full list of configurations for each step of the workflow.

```
define([
  'knockout',
  'jquery',
  'arches',
  'viewmodels/workflow',
  'templates/views/components/plugins/quick-resource-create-workflow.htm',
  // DEFINE EXTRA STEP COMPONENTS HERE AS NEEDED
  'views/components/workflows/final-step'
], function(ko, $, arches, Workflow, quickResourceCreateWorkflowTemplate) {
  return ko.components.register('quick-resource-create-workflow', {
    viewModel: function(params) {
      this.componentName = 'quick-resource-create-workflow';
      this.quitUrl = "/search";
      this.stepConfig = [
        // ADD STEP CONFIG ITEMS HERE
      ];
      Workflow.apply(this, [params]);
    },
    template: quickResourceCreateWorkflowTemplate,
  });
});
```

Workflow Step Configs

Now let's look at what one of these `stepConfig` items should look like. At minimum, it will have the following properties:

title

This will appear in the tab for the step.

name

An internal id for this workflow step that may be referenced by later steps, for example `'initial-step'`. *This value must be unique across all other steps in the workflow.*

required

Use `true` for `false` to determine whether this step must be completed by the user before moving on to the next one.

workflowstepclass

The class for this step. *(Need more info here)*

informationboxdata

The information box gives users guidance on how to complete the workflow step, and must consist of **heading** and **text** elements (see example below).

layoutSections

A list of the sections that appear within this step. These items will be covered in more detail next.

Put together, a `stepConfig` will look something like this:

```
{
  title: 'Create Historic Resource',
  name: 'set-basic-info',
  required: true,
  workflowstepclass: 'create-project-project-name-step',
  informationboxdata: {
    heading: 'Create historic resource here',
    text: 'Begin by providing the name and type of historic resource you are adding,
↪ to the database.',
  },
  layoutSections: [
    // ADD LAYOUT SECTIONS HERE
  ]
}
```

Other properties may be present in a step config if they are needed for more complex workflows.

Layout Sections and Component Configs

A workflow step can have one or more `layoutSections`, each of which contains a list of `componentConfigs`. Component configs are where we reference the part of the resource model that we want users to access. Simple workflows like our example can use multiple component configs that point to different nodegroups, but more complex steps will typically have only one layout section with one component config, the latter ultimately pointing to a custom *step component*.

```
layoutSections: [
  {
    componentConfigs: [
```

(continues on next page)

(continued from previous page)

```
        // INSERT COMPONENT CONFIGS HERE
    ],
}
]
```

The properties of a component config are as follows:

componentName

The id of the UI component that will be used to render this piece of the step. This can be 'default-card' to use Arches' default display. However, you can also write workflow-specific *step components* to handle more complex behavior, and this is where you would reference them. (Any custom step components used here must be added to the define list at the top of the file.)

uniqueInstanceName

An id by which this component can be referenced. *This value must be unique across all other component configs in the step.*

tilesManaged

This must be 'none', 'one', or 'multi', and it determines how many new tiles will be created with this component. Even if the card has a cardinality > 1 in the resource model, setting 'one' here will still disallow multiple values from being created.

parameters

These parameters will be passed to the component. Typically, in the first step you will only use `graphid` (for the resource model) and `nodegroupid` (to determine which nodegroup/card to show). Later steps will also need to be passed the `resourceid` which is pulled from the first step. *Keep in mind that custom step components may require extra parameters.*

```
{
  componentName: 'default-card',
  uniqueInstanceName: 'resource-name', /* unique to step */
  tilesManaged: 'one',
  parameters: {
    graphid: '99417385-b8fa-11e6-84a5-026d961c88e6',
    nodegroupid: '574b58a3-e747-11e6-84a6-026d961c88e6',
  }
}
```

In this example, `graphid` refers to the UUID for the Heritage Resource Model, and `nodegroupid` is the UUID for the nodegroup that holds the Name and Name Type nodes.

Note: There are a couple of ways to find the `nodegroupid`.

1. In the Arches web UI, open the graph designer for the resource model and use your browser's dev tools to isolate the element for the nodegroup you want. The UUID will be visible in the HTML.
2. Using the Django shell:

```
python manage.py shell

>>> from arches.app.models.models import Node
>>> Node.objects.get(name="Name", graph__name="Heritage Resource Model").pk
```

In the second step of our example workflow, where the user will enter a keyword for the new resource, we'll need to pass an extra parameter `resourceid` that was created in the first step. Doing so looks like this:

```
parameters: {
  graphid: '99417385-b8fa-11e6-84a5-026d961c88e6',
  nodegroupid: '3d919f0d-e747-11e6-84a6-026d961c88e6', // UUID for the Keyword
  ↪nodegroup
  resourceid: "['set-basic-info']['resource-name'][0]['resourceInstanceId']",
}
```

To break this resourceid entry down:

- 'set-basic-info' is the **name** of the step from which we are pulling the id (see our first step above)
- 'resource-name' is the **uniqueInstanceName** of the component config in which the tile was created
- 0 is the first tile object
- 'resourceInstanceId' is the property of the tile that we are looking for

Patterns like this can be used elsewhere within workflows to pass information from step to step.

The Final Step

The final step of our example workflow looks like this:

```
{
  title: 'Finish',
  name: 'add-resource-complete', /* unique to workflow */
  description: 'Finish the resource creation.',
  layoutSections: [
    {
      componentConfigs: [
        {
          componentName: 'final-step',
          uniqueInstanceName: 'create-resource-final',
          tilesManaged: 'none',
          parameters: {
            resourceid: "['set-basic-info']['resource-name'][0][
  ↪'resourceInstanceId']",
          },
        ],
      ],
    ],
  ],
}
```

As you can see, no tiles are created here, and we are using the default 'final-step' component that Arches provides (you'll note this component is defined at the top of the file). This step will contain a save/cancel prompt.

Workflows often contain more elaborate final steps than the default one presented here, for example you may want to list all of the data that has been entered throughout the workflow so the user can review it before saving. This behavior is not available by default, but here is an example of a final step with that capability:

- [Step config](#)
- [Step component JS](#)
- [Step component HTML](#)

As you can see, the component gathers all data for the resource and markdown simply hard-codes the presentation of each individual node.

Full Example Workflow

Putting it all together, our main workflow component looks like this:

```
define([
  'knockout',
  'jquery',
  'arches',
  'viewmodels/workflow',
  'templates/views/components/plugins/quick-resource-create-workflow.htm',
  'views/components/workflows/final-step'
], function(ko, $, arches, Workflow, quickResourceCreateWorkflowTemplate) {
  return ko.components.register('quick-resource-create-workflow', {
    viewModel: function(params) {
      this.componentName = 'quick-resource-create-workflow';
      this.quitUrl = "/search";
      this.stepConfig = [
        {
          title: 'Create Historic Resource',
          name: 'set-basic-info', /* unique to workflow */
          required: true,
          workflowstepclass: 'create-project-project-name-step',
          informationboxdata: {
            heading: 'Create historic resource here',
            text: 'Begin by providing the name and type of historic resource_
↪you are adding to the database.',
          },
          layoutSections: [
            {
              componentConfigs: [
                {
                  componentName: 'default-card',
                  uniqueInstanceName: 'resource-name', /* unique to_
↪step */
                  tilesManaged: 'one',
                  parameters: {
                    graphid: '99417385-b8fa-11e6-84a5-026d961c88e6',
                    nodegroupid: '574b58a3-e747-11e6-84a6-
↪026d961c88e6',
                  },
                },
              ],
            },
            {
              componentConfigs: [
                {
                  componentName: 'default-card',
                  uniqueInstanceName: 'resource-type', /* unique to_
↪step */
```

(continues on next page)

(continued from previous page)

```

        tilesManaged: 'one',
        parameters: {
            graphid: '99417385-b8fa-11e6-84a5-026d961c88e6',
            nodegroupid: '620aac67-e747-11e6-84a6-
↪026d961c88e6',
        },
    },
],
},
]
},
{
    title: 'Add Keywords',
    name: 'add-keywords', /* unique to workflow */
    required: false,
    informationboxdata: {
        heading: 'Add a keyword',
        text: 'Optionally add keywords to this historic resource.',
    },
    layoutSections: [
        {
            componentConfigs: [
                {
                    componentName: 'default-card',
                    uniqueInstanceName: 'resource-keywords', /* unique_
↪to step */
                    tilesManaged: 'one',
                    parameters: {
                        graphid: '99417385-b8fa-11e6-84a5-026d961c88e6',
                        nodegroupid: '3d919f0d-e747-11e6-84a6-
↪026d961c88e6',
                        resourceid: "['set-basic-info']['resource-name
↪']['[0]['resourceInstanceId']",
                    },
                },
            ],
        },
    ],
},
{
    title: 'Finish',
    name: 'add-resource-complete', /* unique to workflow */
    description: 'Finish the resource creation.',
    layoutSections: [
        {
            componentConfigs: [
                {
                    componentName: 'final-step',
                    uniqueInstanceName: 'create-resource-final',
                    tilesManaged: 'none',
                    parameters: {
                        resourceid: "['set-basic-info']['resource-name

```

(continues on next page)

(continued from previous page)

```
↪ '')[0]['resourceInstanceId']",
    },
    },
  ],
  },
],
}
];
Workflow.apply(this, [params]);
},
template: quickResourceCreateWorkflowTemplate,
});
});
```

Step Components

You may want to create custom components for your workflow steps to handle more complex data entry tasks. These should be stored in a `workflow` directory, or grouped into subdirectories thematically. A step component can be used by any workflow, as long as it is passed the correct parameters.

Important: If you are loading a package with a workflow in it, you will need to manually copy step component files into your project—they are not handled by the package load process.

Here are some examples of workflows that use custom step components you can look at when beginning to construct your own:

- **Arches HER Workflows**
 - All files here define workflows besides the one called `init-workflow.js` (which is a standard plugin)
 - In one of these workflows, find a `componentName` that is *not* `'default-card'`
 - **Now, look for that component name in the following two places:**
 - * [Arches HER Step Components \(JS\)](#)
 - * [Arches HER Step Components \(HMTL\)](#)

Registering your Workflow

After placing your workflow files in the proper directories within your project, you are ready to register it. See [Plugin Commands](#) for more information.

Accessing the Workflow

If the Workflow (or any other Plugin) is registered but is not visible, an administrator must grant access to it via the Django admin panel on a per-user or per-group basis.

Navigate to `localhost:8000/admin` and login, and locate profiles for the user(s) or group(s) that should be able to access the Workflow. Find the “User/Group permissions” section, scroll to your workflow, and add the “view” privilege. Click “SAVE” to finish.

- *Card Components*

- Used to create modular data entry and data display units that can be nested arbitrarily in the UI. Creating a custom card component can provide a more complex UI.

- **Structure**

```
my_project/cards/my-card.json          <-- Registration
↳ config
my_project/media/js/views/components/cards/my-card.js      <-- UI
↳ component
my_project/templates/views/components/cards/my-card.htm    <-- UI
↳ component
```

- *Datatypes*

- Allows you to store new types of data in Arches, and attach custom logic (like save operations or storage techniques) to that datatype.

- **Structure**

```
my_project/datatypes/my_datatype.py    <-- Main logic (and registration
↳ config)
my_project/media/js/views/datatypes/my-datatype.js        <-- UI
↳ component
my_project/templates/views/datatypes/widgets/my-datatype.htm <-- UI
↳ component
```

- *Functions*

- Can be triggered whenever a tile is saved, providing a framework for the introduction of Python code into day-to-day operations.

- **Structure**

```
my_project/functions/my_function.py     <-- Main logic (and
↳ registration config)
my_project/media/js/views/functions/my-function.js       <--
↳ UI component
my_project/templates/views/functions/widgets/my-function.htm <--
↳ UI component
```

- *Plugins*

- Provides a generic framework for adding new web pages to your project.

- **Structure**

```
my_project/plugins/my-plugin.json          <-- Registration_
↳ config
my_project/media/js/views/components/plugins/my-plugin.js  <-- UI_
↳ component
my_project/templates/views/components/plugins/my-plugin.htm <-- UI_
↳ component
```

- *Resource Reports*

- Can be customized to augment the way a resource instance is presented to the site users and the public.
- **Structure**

```
my_project/reports/my-report.json          <--_
↳ Registration config
my_project/media/js/views/components/report-templates/my-report.js  _
↳ <-- UI component
my_project/templates/views/components/report-templates/my-report.htm _
↳ <-- UI component
```

- *Search Filters*

- Use these to add extra search capabilities to the interface, or to inject extra filters behind the scenes.
- **Structure**

```
my_project/search/my_filter.py            <-- Main logic (and registration_
↳ config)
my_project/media/js/views/components/search/my-filter.js  <-- UI_
↳ component
my_project/templates/views/components/search/my-filter.htm <-- UI_
↳ component
```

- *Widgets*

- Widgets allow you to customize how data entered into Arches, and how that data is presented to the public.
- **Structure**

```
my_project/widgets/my-widget.json          <-- Registration_
↳ config
my_project/media/js/views/components/widgets/my-widget.js  <-- UI_
↳ component
my_project/templates/views/components/widgets/my-widget.htm <-- UI_
↳ component
```

- *Workflows*

- Workflows are a special type of Plugin that allow you to abstract the data entry process away from the default graph tree interface into a step-through set of pages.
- **Basic Structure**

```
my_project/plugins/my-workflow.json          <--_
↳ Registration config
my_project/media/js/views/components/plugins/my-workflow.js  <--_
```

(continues on next page)

(continued from previous page)

```

↪Main UI component
my_project/templates/views/components/plugins/my-workflow.htm <--↪
↪Main UI component

```

– Custom Step Components

```

my_project/media/js/views/components/workflows/my-workflow-component.
↪js      <-- Step component
my_project/templates/views/components/workflows/my-workflow-component.
↪htm     <-- Step component

```

Extension Architecture

Though there is some variation across extension types, Arches does use a common architecture pattern to construct extensions. Generally speaking, the user interface for the extension exists in a new **component** (JS/HTML), and any backend code (if applicable) will be in a **module** (Python). Initial configuration details will be stored in **json**, either in a standalone file or at the top of a module.

A component

All extensions are expected to have some sort of user interface, and this is created with a pair of files: one HTML template (.htm) and one JavaScript file (.js). Components are constructed with [KnockoutJS](#).

These files must live here (using a widget as an example):

```

my_project/media/js/views/components/widgets/custom-widget.js
my_project/templates/views/components/widgets/custom-widget.htm

```

A JSON configuration file

A .json file will store a set of initial configuration details about the extension, which are loaded into the database when the extension is registered. This file is only used during registration.

These files typically live here:

```

my_project/widgets/custom-widget.json

```

A module

A few extension types, like Functions, are written in Python. For these, a .py module must be supplied. Instead of a JSON configuration file, initial configs are stored at the beginning of the module in a dictionary named **details**.

A module's location follows the same pattern as JSON configuration files:

```

my_project/datatypes/custom_function.py

```

Extension Data Models

The backend models for each extension type are shown below.

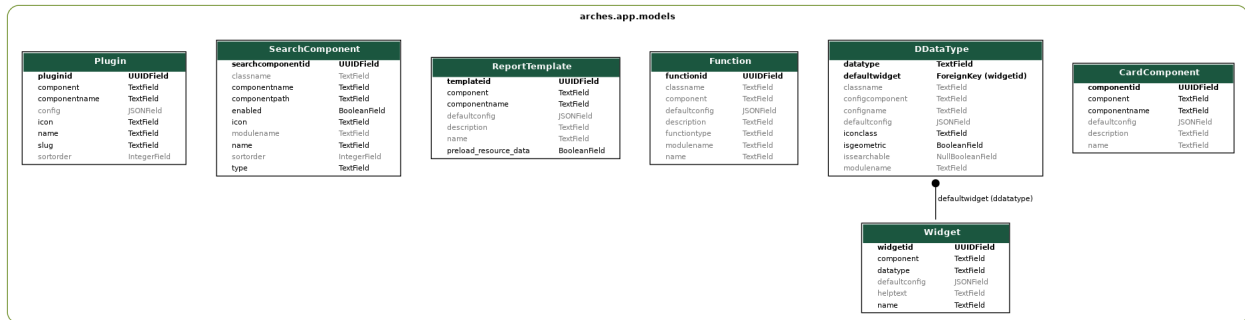


Fig. 38: Each extension type is stored in a database table for its kind. *Workflows* are technically *Plugins* so there is no separate table for them.

Managing Extensions

Extensions are “registered” and “unregistered” in one of two ways:

- 1) Via the CLI
- 2) As part of a package load

For CLI commands, see the end of each extension type’s documentation, or checkout the [Command Line Reference](#) page. Checkout [Understanding Packages](#) for more about how and where extensions are stored within packages.

Adding JavaScript Dependencies

This section will help if you are creating an extension that requires a new JavaScript library to be available to your component.

By default, the only dependency in a new project’s package.json file is Arches itself:

```
{
  "name": "my_project",
  "dependencies": {
    "arches": "archesproject/arches#stable/6.0.1"
  }
}
```

This means that when you run `yarn install` on this file, all dependencies from the corresponding branch of the core Arches repo will be installed (in this example, `package.json` from `stable/6.0.1`).

To add a new package, you just need to run `yarn add <package name>` in your project. This will install the new package and update your `package.json` file accordingly.

For example, to add `OpenLayers`, enter the `my_project` directory and run `yarn add ol`. Your `package.json` will now look something like:

```
{
  "name": "my_project",
  "dependencies": {
    "arches": "archesproject/arches#stable/6.0.1",
    "ol": "^6.12.0"
  }
}
```

If you are developing a project, keep track of which version of Arches you are developing against and make sure it is properly reflected in your `package.json`.

Note: When you register a new extension there is no way to automaticate the installation of a new JS dependency, so you'll need to manually run `yarn add` as described above.

Creating New Map Layers

A developer can add new layers to the map by registering them through the command line interface.

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.

Note: By default, new map layers are designated as Overlays. To designate the layer as a Basemap, just add `-b` to the load commands shown below.

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. You can find WMS examples in the [arches4-geo-examples](#) repo.

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.

Adding Click and Hover Styles

In addition to making overlay features selectable, you can define styles for their hover and click states.

1. To do so, each feature in your overlay needs a unique `_featureid`. If you're overlay served from PostGIS, you can define this property in the layer config's `queries` array like so:

```
"queries": [  
  "select gid as __id__, gid as _featureid, site_name, feature_info_content, st_  
  ↪asgeojson(geom) as geojson, st_transform(geom, 900913) as __geometry__ from_  
  ↪example_layer"  
]
```

2. Next you will need to ensure your *source-layer* is properly defined. In the source layer the *source-layer* property must match the *id* property and cannot contain spaces or periods. This layer will be hidden when the hover or click layer is revealed, so this should be a fill layer if your click or hover layers contain a fill.
3. Define the hover and click layers. These each must have a `_featureid` filter their ids must be suffixed with either a *-click* or *-hover*. For example:

```
{  
  "layout": {  
    "visibility": "visible"  
  },  
  "source": "example_layer",  
  "filter": [  
    "all",  
    [  
      "=",  
      "$type",  
      "Polygon"  
    ],  
    [  
      "=",  
      "_featureid",  
      ""  
    ]  
  ]  
}
```

(continues on next page)

(continued from previous page)

```

    ],
    "paint": {
      "fill-color": "rgb(0, 255, 0)",
      "fill-opacity": 0.5
    },
    "source-layer": "example_layer",
    "type": "fill",
    "id": "example_layer-hover"
  }
}

```

4. If you are loading your layers from a package, each layer must have an accompanying *meta.json* file with a name defined. This will ensure that the *source-layer* property is saved to the layer as you intend. If you do not have a *meta.json* file, the source-layer name will be the map layer's file name, and will probably not work properly. See the example package for an example:

https://github.com/archesproject/arches4-example-pkg/tree/master/map_layers/tile_server/overlays/vector_example

Customizing Map Popup Content

You can display custom HTML in the search map popup when a user hovers or clicks on a feature in a vector layer.

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 "feature_info_content".
3. Populate this property with either an html element or a url from which to load html. If you use a url, you will need to update the 'ALLOWED_POPUP_HOSTS' to include the host from which you want to request HTML.
4. Add the overlay as you would any tileserver layer (see above).

You will now be able to add this layer to the map see the markup defined in the 'feature_info_content' in the search map popup.

Creating HTML Export Templates

The HTML export templates are used to allow search results in Arches to be exported as a set of html files containing formatted report styled documents.

They are designed to be read offline by embedding a small CSS framework called Milligram, along with some additional elements.

The templates must be written by a developer and added to the Arches project template folder.

There can only be one HTML export template for each model in the application.

Template Location

The templates should be saved to a folder in `/path/to/project_workspace/project_name/project_name/templates/html_export`.

The templates must have the name format of `<graph id>.htm`, with the `<graph id>` value being the model's UUID.

example: `076f9381-7b00-11e9-8d6b-80000b44d1d9.htm`

Templating Language

The templates are using the native Django templating engine, information of which can be found here: <https://docs.djangoproject.com/en/stable/ref/templates/language/>

Resources Context Data

When the export process loads the template ready for it to be rendered, it will be passed a `resources` context object that contains the data for the resources to be written. This list is then iterated to build each record.

The structure of each resources object is as below:

```
{
  "displaydescription": " Excavation by Department Of Greater London Archaeology, April
↳to May 1990, found a large 'soft spot' which was either a quarry ditch or was dug for
↳dumping waste. Also uncovered features of 20th century date relating to a building
↳called Green Acres.",
  "displayname": "Open Area Excavation at Lichfield Gardens",
  "graph_id": "b9e0701e-5463-11e9-b5f5-000d3ab1e588",
  "legacyid": "06eb7a47-baf7-4c79-aeab-2ffabe2502ea",
  "map_popup": "<Activity Descriptions>",
  "resource": {
    "...": "..."
  },
  "resourceinstanceid": "06eb7a47-baf7-4c79-aeab-2ffabe2502ea"
}
```

Some id and display information is directly accessible from the object, such as the `displayname` and `displaydescription`. The rest of the resource data is contained in the `resource` dictionary as a “disambiguated” version of the resource.

The dict uses the branch/card/node names as the keys, with the `@display_value` key containing the presentation value for that node. There are other values included in the dicts that can be used to add richer functionality if needed.

Note: This structure closely matches the JSON produced when looking at a resource when specifying the `format=json&v=beta`

```
curl http://localhost:8000/resources/<resourceinstanceid>?format=json&v=beta
```

Ensure that you use the `v=beta` parameter as the functionality is using this version of the data formatting.

```
"resource": {
  "Descriptions": [{
```

(continues on next page)

(continued from previous page)

```

    "Description": {
        "@display_value": "Amendment date:none"
    },
    "Description Language": {
        "Description Language Metatype": null
    },
    "Description Type": {
        "@display_value": "Notes",
        "Description Metatype": null,
        "concept_id": "f1cbae8f-0090-47dc-8252-ee533a2deb29",
        "language_id": "en",
        "value": "Notes",
        "valueid": "daa4cddc-8636-4842-b836-eb2e10aabe18",
        "valuetype_id": "prefLabel"
    }
  },
  "Designation and Protection Assignment": [],
  "Heritage Area Names": [],
  "Location Data": {},
  "System Reference Numbers": {}
}

```

Custom Template Filters

The resources context data sent to the templates has an incompatibility with the standard Django dot notation for accessing values. Usually you would access a dictionary value using `dict.key`, but the resource dictionary uses keys with spaces that can't be parsed `resource_data.Activity Names`.

The other difficulty is that the resource dictionary may not contain the key you are looking for if no tiles for that data exists. Therefore, you need to check for the existence of the key before you access it.

To solve this, two new template filters were added to `arches/arches/templatetags/template_tags.py`:

has_key

You can use `has_key` as part of an `if` tag to check if there is a key in the object. If you try to access the object without checking then it may error should the key not be present.

```

{% if asset_names|has_key:"Asset Name Use Type" %}
    {# you can access without error asset_names["Asset Name Use Type"] #}
{% endif %}

```

val_from_key

This function allows you to retrieve a value from a key that is not Django templating compliant. These can be chained to access nested dictionaries (careful that you are sure the nested dictionary exists).

```

<h2>{{ asset_names|val_from_key:"Asset Name"|val_from_key:"@display_value" }}</h2>

```

json_to_obj

This function can be used in the rare instance where the value is JSON and you need to convert that to a dict or list in order to access the values.

```
<a href="{{ external_source|val_from_key:'URL'|val_from_key:'@display_value'|json_to_obj|val_from_key:'url' }}" target="_blank">Document Link</a>
```

example combining has_key and val_from_key.

```
{% if asset_names|has_key:"Asset Name Use Type" %}
  <h2>{{ asset_names|val_from_key:"Asset Name"|val_from_key:"@display_value" }}</h2>
{% endif %}
```

example of using chained filters to access nested values

```
<strong>Primary Reference Number: </strong>{{ resource_data|val_from_key:"System_
Reference Numbers"|val_from_key:"PrimaryReferenceNumber"|val_from_key:"Primary_
Reference Number"|val_from_key:"@display_value" }}
```

example transforming a url datatype value into a dict for use within a data table call

```
<div class="rcell" data-title="URL">
  {% with exref|val_from_key:"URL"|val_from_key:"@display_value"|json_to_obj as URL_
Dict %}
    {% if URL_Dict|has_key:"url" and URL_Dict|has_key:"url_label" %}
      <a href="{{ URL_Dict|val_from_key:'url' }}">{{ URL_Dict|val_from_key:'url_label'
}}</a>
    {% else %}
      <br />
    {% endif %}
  {% endwith %}
</div>
```

Basic Template

The basic template below will provide the CSS framework, add the custom template tags (used to add custom functions to the template engine), and the initial resource loop within which to start your document.

```
{% load template_tags %}
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>--ADD DOCUMENT TITLE--</title>
    <style>
      /* Milligram css */
      *,*:after,*:before{box-sizing:inherit}html{box-sizing:border-box;font-size:62.5%}
body{color:#606c76;font-family:'Roboto', 'Helvetica Neue', 'Helvetica', 'Arial', sans-
serif;font-size:1.6em;font-weight:300;letter-spacing:.01em;line-height:1.6}blockquote
{border-left:.3rem solid #d1d1d1;margin-left:0;margin-right:0;padding:1rem 1.5rem}
blockquote *:last-child{margin-bottom:0}.button,button,input[type='button'],input[
type='reset'],input[type='submit']{background-color:#9b4dca;border:.1rem solid #9b4dca;
border-radius:.4rem;color:#fff;cursor:pointer;display:inline-block;font-size:1.1rem;
font-weight:700;height:3.8rem;letter-spacing:.1rem;line-height:3.8rem;padding:0 3.0rem;
```

(continues on next page)

(continued from previous page)

```

→text-align:center;text-decoration:none;text-transform:uppercase;white-space:nowrap}.
→button:focus,.button:hover,button:focus,button:hover,input[type='button']:focus,
→input[type='button']:hover,input[type='reset']:focus,input[type='reset']:hover,
→input[type='submit']:focus,input[type='submit']:hover{background-color:#606c76;border-
→color:#606c76;color:#fff;outline:0}.button[disabled],button[disabled],input[type=
→'button'][disabled],input[type='reset'][disabled],input[type='submit'][disabled]
→{cursor:default;opacity:.5}.button[disabled]:focus,.button[disabled]:hover,
→button[disabled]:focus,button[disabled]:hover,input[type='button'][disabled]:focus,
→input[type='button'][disabled]:hover,input[type='reset'][disabled]:focus,input[type=
→'reset'][disabled]:hover,input[type='submit'][disabled]:focus,input[type='submit
→'] [disabled]:hover{background-color:#9b4dca;border-color:#9b4dca}.button.button-
→outline,button.button-outline,input[type='button'].button-outline,input[type='reset'].
→button-outline,input[type='submit'].button-outline{background-color:transparent;color:
→#9b4dca}.button.button-outline:focus,.button.button-outline:hover,button.button-
→outline:focus,button.button-outline:hover,input[type='button'].button-outline:focus,
→input[type='button'].button-outline:hover,input[type='reset'].button-outline:focus,
→input[type='reset'].button-outline:hover,input[type='submit'].button-outline:focus,
→input[type='submit'].button-outline:hover{background-color:transparent;border-color:
→#606c76;color:#606c76}.button.button-outline[disabled]:focus,.button.button-
→outline[disabled]:hover,button.button-outline[disabled]:focus,button.button-
→outline[disabled]:hover,input[type='button'].button-outline[disabled]:focus,input[type=
→'button'].button-outline[disabled]:hover,input[type='reset'].button-
→outline[disabled]:focus,input[type='reset'].button-outline[disabled]:hover,input[type=
→'submit'].button-outline[disabled]:focus,input[type='submit'].button-
→outline[disabled]:hover{border-color:inherit;color:#9b4dca}.button.button-clear,button.
→button-clear,input[type='button'].button-clear,input[type='reset'].button-clear,
→input[type='submit'].button-clear{background-color:transparent;border-
→color:transparent;color:#9b4dca}.button.button-clear:focus,.button.button-clear:hover,
→button.button-clear:focus,button.button-clear:hover,input[type='button'].button-
→clear:focus,input[type='button'].button-clear:hover,input[type='reset'].button-
→clear:focus,input[type='reset'].button-clear:hover,input[type='submit'].button-
→clear:focus,input[type='submit'].button-clear:hover{background-color:transparent;
→border-color:transparent;color:#606c76}.button.button-clear[disabled]:focus,.button.
→button-clear[disabled]:hover,button.button-clear[disabled]:focus,button.button-
→clear[disabled]:hover,input[type='button'].button-clear[disabled]:focus,input[type=
→'button'].button-clear[disabled]:hover,input[type='reset'].button-
→clear[disabled]:focus,input[type='reset'].button-clear[disabled]:hover,input[type=
→'submit'].button-clear[disabled]:focus,input[type='submit'].button-
→clear[disabled]:hover{color:#9b4dca}code{background:#f4f5f6;border-radius:.4rem;font-
→size:86%;margin:0 .2rem;padding:.2rem .5rem;white-space:nowrap}pre{background:#f4f5f6;
→border-left:0.3rem solid #9b4dca;overflow-y:hidden}pre>code{border-radius:0;
→display:block;padding:1rem 1.5rem;white-space:pre}hr{border:0;border-top:0.1rem solid
→#f4f5f6;margin:3.0rem 0}input[type='color'],input[type='date'],input[type='datetime'],
→input[type='datetime-local'],input[type='email'],input[type='month'],input[type='number
→'],input[type='password'],input[type='search'],input[type='tel'],input[type='text'],
→input[type='url'],input[type='week'],input:not([type]),textarea,select{-webkit-
→appearance:none;background-color:transparent;border:0.1rem solid #d1d1d1;border-
→radius:.4rem;box-shadow:none;box-sizing:inherit;height:3.8rem;padding:.6rem 1.0rem .
→7rem;width:100%}input[type='color']:focus,input[type='date']:focus,input[type='datetime
→']:focus,input[type='datetime-local']:focus,input[type='email']:focus,input[type='month
→']:focus,input[type='number']:focus,input[type='password']:focus,input[type='search
→']:focus,input[type='tel']:focus,input[type='text']:focus,input[type='url']:focus,

```

(continues on next page)

(continued from previous page)

```

→input[type='week']:focus,input:not([type]):focus,textarea:focus,select:focus{border-
→color:#9b4dca;outline:0}select{background:url('data:image/svg+xml;utf8,<svg xmlns=
→"http://www.w3.org/2000/svg" viewBox="0 0 30 8" width="30"><path fill="%23d1d1d1" d=
→"M0,0l6,8l6-8"></path></svg>') center right no-repeat;padding-right:3.0rem}select:focus
→{background-image:url('data:image/svg+xml;utf8,<svg xmlns="http://www.w3.org/2000/svg"
→viewBox="0 0 30 8" width="30"><path fill="%239b4dca" d="M0,0l6,8l6-8"></path></svg>')}
→select[multiple]{background:none;height:auto}textarea{min-height:6.5rem}label,legend
→{display:block;font-size:1.6rem;font-weight:700;margin-bottom:.5rem}fieldset{border-
→width:0;padding:0}input[type='checkbox'],input[type='radio']{display:inline}.label-
→inline{display:inline-block;font-weight:normal;margin-left:.5rem}.container{margin:0
→auto;max-width:112.0rem;padding:0 2.0rem;position:relative;width:100%}.row
→{display:flex;flex-direction:column;padding:0;width:100%}.row.row-no-padding{padding:0}
→.row.row-no-padding>.column{padding:0}.row.row-wrap{flex-wrap:wrap}.row.row-top{align-
→items:flex-start}.row.row-bottom{align-items:flex-end}.row.row-center{align-
→items:center}.row.row-stretch{align-items:stretch}.row.row-baseline{align-
→items:baseline}.row .column{display:block;flex:1 1 auto;margin-left:0;max-width:100%;
→width:100%}.row .column.column-offset-10{margin-left:10%}.row .column.column-offset-20
→{margin-left:20%}.row .column.column-offset-25{margin-left:25%}.row .column.column-
→offset-33,.row .column.column-offset-34{margin-left:33.3333%}.row .column.column-
→offset-40{margin-left:40%}.row .column.column-offset-50{margin-left:50%}.row .column.
→column-offset-60{margin-left:60%}.row .column.column-offset-66,.row .column.column-
→offset-67{margin-left:66.6666%}.row .column.column-offset-75{margin-left:75%}.row .
→column.column-offset-80{margin-left:80%}.row .column.column-offset-90{margin-left:90%}.
→row .column.column-10{flex:0 0 10%;max-width:10%}.row .column.column-20{flex:0 0 20%;
→max-width:20%}.row .column.column-25{flex:0 0 25%;max-width:25%}.row .column.column-33,
→.row .column.column-34{flex:0 0 33.3333%;max-width:33.3333%}.row .column.column-40
→{flex:0 0 40%;max-width:40%}.row .column.column-50{flex:0 0 50%;max-width:50%}.row .
→column.column-60{flex:0 0 60%;max-width:60%}.row .column.column-66,.row .column.column-
→67{flex:0 0 66.6666%;max-width:66.6666%}.row .column.column-75{flex:0 0 75%;max-
→width:75%}.row .column.column-80{flex:0 0 80%;max-width:80%}.row .column.column-90
→{flex:0 0 90%;max-width:90%}.row .column .column-top{align-self:flex-start}.row .
→column .column-bottom{align-self:flex-end}.row .column .column-center{align-
→self:center}@media (min-width: 40rem){.row{flex-direction:row;margin-left:-1.0rem;
→width:calc(100% + 2.0rem)}.row .column{margin-bottom:inherit;padding:0 1.0rem}}a{color:
→#9b4dca;text-decoration:none}a:focus,a:hover{color:#606c76}dl,ol,ul{list-style:none;
→margin-top:0;padding-left:0}dl dl,dl ol,dl ul,ol dl,ol ol,ol ul,ul dl,ul ol,ul ul{font-
→size:90%;margin:1.5rem 0 1.5rem 3.0rem}ol{list-style:decimal inside}ul{list-
→style:circle inside}.button,button,dd,dt,li{margin-bottom:1.0rem}fieldset,input,select,
→textarea{margin-bottom:1.5rem}blockquote,dl,figure,form,ol,p,pre,table,ul{margin-
→bottom:2.5rem}table{border-spacing:0;display:block;overflow-x:auto;text-align:left;
→width:100%}td,th{border-bottom:0.1rem solid #e1e1e1;padding:1.2rem 1.5rem}td:first-
→child,th:first-child{padding-left:0}td:last-child,th:last-child{padding-right:0}@media
→(min-width: 40rem){table{display:table;overflow-x:initial}}b,strong{font-weight:bold}p
→{margin-top:0}h1,h2,h3,h4,h5,h6{font-weight:300;letter-spacing:-.1rem;margin-bottom:2.
→0rem;margin-top:0}h1{font-size:4.6rem;line-height:1.2}h2{font-size:3.6rem;line-
→height:1.25}h3{font-size:2.8rem;line-height:1.3}h4{font-size:2.2rem;letter-spacing:-.
→08rem;line-height:1.35}h5{font-size:1.8rem;letter-spacing:-.05rem;line-height:1.5}h6
→{font-size:1.6rem;letter-spacing:0;line-height:1.4}img{max-width:100%}.clearfix:after
→{clear:both;content:' ';display:table}.float-left{float:left}.float-right{float:right}
/* General */
html{font-size:55%}body{color:#000}.section-title blockquote{border:.3rem solid
→#d1d1d1;background-color:#90C0D8}h3{margin-bottom:.2rem}.container{margin:0;max-

```

(continues on next page)

(continued from previous page)

```

width:100%;hr{border-top:.3rem solid #d1d1d1;margin:2rem 0}ul{list-style:none}.
location-details li{margin-bottom:0}@media print{.section-title:not(:first-child){page-
break-before:always}.keepstogether{break-inside:avoid}}
/* Responsive tables */
.rtable{margin:0 0 40px 0;width:100%;box-shadow:0 1px 3px rgba(0,0,0,.2);
display:table}@media screen and (max-width:580px){.rtable{display:block}}.rrow
{display:table-row}.rrow:nth-of-type(odd){background-color:#fff}.rrow.rheader{font-
weight:600;background:#d1d1d1}@media screen and (max-width:1024px){.rrow{padding:14px
0 7px;display:block;border-bottom:1px solid #d1d1d1}.rrow.rheader{padding:0;height:6px}
.rrow.rheader .rcell{display:none}.rrow .rcell{margin-bottom:10px;border:none}.rrow .
rcell:before{margin-bottom:3px;content:attr(data-title);min-width:98px;font-size:.85em;
line-height:10px;font-weight:700;text-transform:uppercase;display:block}.rcell
{padding:6px 12px;display:table-cell;border-bottom:1px solid #d1d1d1}.rcell ul{list-
style:none}.rcell li{margin-bottom:0}@media screen and (max-width:1024px){.rcell
{padding:2px 16px;display:block}}@media print{html{font-size:40%}.rtable{box-
shadow:none;border:1px solid #d1d1d1}.rcell{border:1px solid #d1d1d1}.location-details.
row{flex-direction:row}}
</style>
</head>
<body>
<header>
<h1>--ADD MAIN HEADER TITLE--</h1>
</header>
<main>
{% for resource in resources %}
<h2>{{ resource.displayname }}</h2>
<p>{{ resource.displaydesicription }}</p>
{% with resource_data=resource.resource %}
... build template
{% endwhile %}
{% endfor %}
</main>
</body>
</html>

```

Below show examples of how you can fetch specific data out of the resource object to build a section of the document.

```

{% with resource_data=resource.resource %}
<section class="section-title">
  <blockquote>
    {% if resource_data|has_key:"Heritage Asset Names" %}
    {% for n in resource_data|val_from_key:"Heritage Asset Names" %}
    {% if n|has_key:"Asset Name Use Type" %}
    {% if n|val_from_key:"Asset Name Use Type"|val_from_key:"@display_value"
    == "Primary" %}
    <h2>{{ n|val_from_key:"Asset Name"|val_from_key:"@display_value" }}</
    h2>
    {% endif %}
    {% endif %}
    {% endfor %}
    {% endif%}
  <p>

```

(continues on next page)

(continued from previous page)

```

<strong>Primary Reference Number: </strong>{{ resource_data|val_from_key:
↪ "System Reference Numbers"|val_from_key:"PrimaryReferenceNumber"|val_from_key:"Primary_
↪ Reference Number"|val_from_key:"@display_value" }}<br>
</p>
</blockquote>
</section>
{% endwith %}

```

Advanced Template Example

Below is an example that includes sections that build tables and group elements together.

Note: Use `<div class="keeptogether"></div>` blocks around tables and other iterated sections to force the styling to keep things on the same page where possible when printing.

```

{% load template_tags %}
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Report</title><style>
      /* Milligram css */
      *,*:after,*:before{box-sizing:inherit}html{box-sizing:border-box;font-size:62.5
↪ %}body{color:#606c76;font-family:'Roboto','Helvetica Neue','Helvetica','Arial',
↪ sans-serif;font-size:1.6em;font-weight:300;letter-spacing:.01em;line-height:1.6}
↪ blockquote{border-left:.3rem solid #d1d1d1;margin-left:0;margin-right:0;padding:1rem
↪ 1.5rem}blockquote *:last-child{margin-bottom:0}.button,button,input[type='button'],
↪ input[type='reset'],input[type='submit']{background-color:#9b4dca;border:.1rem solid
↪ #9b4dca;border-radius:.4rem;color:#fff;cursor:pointer;display:inline-block;font-size:1.
↪ 1rem;font-weight:700;height:3.8rem;letter-spacing:.1rem;line-height:3.8rem;padding:0 3.
↪ 0rem;text-align:center;text-decoration:none;text-transform:uppercase;white-
↪ space:nowrap}.button:focus,.button:hover,button:focus,button:hover,input[type='button
↪ ']:focus,input[type='button']:hover,input[type='reset']:focus,input[type='reset
↪ ']:hover,input[type='submit']:focus,input[type='submit']:hover{background-color:
↪ #606c76;border-color:#606c76;color:#fff;outline:0}.button[disabled],button[disabled],
↪ input[type='button'][disabled],input[type='reset'][disabled],input[type='submit
↪ '][disabled]{cursor:default;opacity:.5}.button[disabled]:focus,.button[disabled]:hover,
↪ button[disabled]:focus,button[disabled]:hover,input[type='button'][disabled]:focus,
↪ input[type='button'][disabled]:hover,input[type='reset'][disabled]:focus,input[type=
↪ 'reset'][disabled]:hover,input[type='submit'][disabled]:focus,input[type='submit
↪ '][disabled]:hover{background-color:#9b4dca;border-color:#9b4dca}.button.button-
↪ outline,button.button-outline,input[type='button'].button-outline,input[type='reset'].
↪ button-outline,input[type='submit'].button-outline{background-color:transparent;color:
↪ #9b4dca}.button.button-outline:focus,.button.button-outline:hover,button.button-
↪ outline:focus,button.button-outline:hover,input[type='button'].button-outline:focus,
↪ input[type='button'].button-outline:hover,input[type='reset'].button-outline:focus,
↪ input[type='reset'].button-outline:hover,input[type='submit'].button-outline:focus,

```

(continues on next page)

(continued from previous page)

```

→input[type='submit'].button-outline:hover{background-color:transparent;border-color:
→#606c76;color:#606c76}.button.button-outline[disabled]:focus,.button.button-
→outline[disabled]:hover,button.button-outline[disabled]:focus,button.button-
→outline[disabled]:hover,input[type='button'].button-outline[disabled]:focus,input[type=
→'button'].button-outline[disabled]:hover,input[type='reset'].button-
→outline[disabled]:focus,input[type='reset'].button-outline[disabled]:hover,input[type=
→'submit'].button-outline[disabled]:focus,input[type='submit'].button-
→outline[disabled]:hover{border-color:inherit;color:#9b4dca}.button.button-clear,button.
→button-clear,input[type='button'].button-clear,input[type='reset'].button-clear,
→input[type='submit'].button-clear{background-color:transparent;border-
→color:transparent;color:#9b4dca}.button.button-clear:focus,.button.button-clear:hover,
→button.button-clear:focus,button.button-clear:hover,input[type='button'].button-
→clear:focus,input[type='button'].button-clear:hover,input[type='reset'].button-
→clear:focus,input[type='reset'].button-clear:hover,input[type='submit'].button-
→clear:focus,input[type='submit'].button-clear:hover{background-color:transparent;
→border-color:transparent;color:#606c76}.button.button-clear[disabled]:focus,.button.
→button-clear[disabled]:hover,button.button-clear[disabled]:focus,button.button-
→clear[disabled]:hover,input[type='button'].button-clear[disabled]:focus,input[type=
→'button'].button-clear[disabled]:hover,input[type='reset'].button-
→clear[disabled]:focus,input[type='reset'].button-clear[disabled]:hover,input[type=
→'submit'].button-clear[disabled]:focus,input[type='submit'].button-
→clear[disabled]:hover{color:#9b4dca}code{background:#f4f5f6;border-radius:.4rem;font-
→size:86%;margin:0 .2rem;padding:.2rem .5rem;white-space:nowrap}pre{background:#f4f5f6;
→border-left:.3rem solid #9b4dca;overflow-y:hidden}pre>code{border-radius:0;
→display:block;padding:1rem 1.5rem;white-space:pre}hr{border:0;border-top:.1rem solid
→#f4f5f6;margin:3.0rem 0}input[type='color'],input[type='date'],input[type='datetime'],
→input[type='datetime-local'],input[type='email'],input[type='month'],input[type='number
→'],input[type='password'],input[type='search'],input[type='tel'],input[type='text'],
→input[type='url'],input[type='week'],input:not([type]),textarea,select{-webkit-
→appearance:none;background-color:transparent;border:.1rem solid #d1d1d1;border-
→radius:.4rem;box-shadow:none;box-sizing:inherit;height:3.8rem;padding:.6rem 1.0rem .
→7rem;width:100%}input[type='color']:focus,input[type='date']:focus,input[type='datetime
→']:focus,input[type='datetime-local']:focus,input[type='email']:focus,input[type='month
→']:focus,input[type='number']:focus,input[type='password']:focus,input[type='search
→']:focus,input[type='tel']:focus,input[type='text']:focus,input[type='url']:focus,
→input[type='week']:focus,input:not([type]):focus,textarea:focus,select:focus{border-
→color:#9b4dca;outline:0}select{background:url('data:image/svg+xml;utf8,<svg xmlns=
→"http://www.w3.org/2000/svg" viewBox="0 0 30 8" width="30"><path fill="%23d1d1d1" d=
→"M0,0l6,8l6-8"></path></svg>') center right no-repeat;padding-right:3.0rem}select:focus
→{background-image:url('data:image/svg+xml;utf8,<svg xmlns="http://www.w3.org/2000/svg"
→viewBox="0 0 30 8" width="30"><path fill="%239b4dca" d="M0,0l6,8l6-8"></path></svg>')}
→select[multiple]{background:none;height:auto}textarea{min-height:6.5rem}label,legend
→{display:block;font-size:1.6rem;font-weight:700;margin-bottom:.5rem}fieldset{border-
→width:0;padding:0}input[type='checkbox'],input[type='radio']{display:inline}.label-
→inline{display:inline-block;font-weight:normal;margin-left:.5rem}.container{margin:0
→auto;max-width:112.0rem;padding:0 2.0rem;position:relative;width:100%}.row
→{display:flex;flex-direction:column;padding:0;width:100%}.row.row-no-padding{padding:0}
→.row.row-no-padding>.column{padding:0}.row.row-wrap{flex-wrap:wrap}.row.row-top{align-
→items:flex-start}.row.row-bottom{align-items:flex-end}.row.row-center{align-
→items:center}.row.row-stretch{align-items:stretch}.row.row-baseline{align-
→items:baseline}.row .column{display:block;flex:1 1 auto;margin-left:0;max-width:100%;
→width:100%}.row .column.column-offset-10{margin-left:10%}.row .column.column-offset-20

```

(continues on next page)

(continued from previous page)

```

→{margin-left:20%}.row .column.column-offset-25{margin-left:25%}.row .column.column-
→offset-33,.row .column.column-offset-34{margin-left:33.3333%}.row .column.column-
→offset-40{margin-left:40%}.row .column.column-offset-50{margin-left:50%}.row .column.
→column-offset-60{margin-left:60%}.row .column.column-offset-66,.row .column.column-
→offset-67{margin-left:66.6666%}.row .column.column-offset-75{margin-left:75%}.row .
→column.column-offset-80{margin-left:80%}.row .column.column-offset-90{margin-left:90%}.
→row .column.column-10{flex:0 0 10%;max-width:10%}.row .column.column-20{flex:0 0 20%;
→max-width:20%}.row .column.column-25{flex:0 0 25%;max-width:25%}.row .column.column-33,
→.row .column.column-34{flex:0 0 33.3333%;max-width:33.3333%}.row .column.column-40
→{flex:0 0 40%;max-width:40%}.row .column.column-50{flex:0 0 50%;max-width:50%}.row .
→column.column-60{flex:0 0 60%;max-width:60%}.row .column.column-66,.row .column.column-
→67{flex:0 0 66.6666%;max-width:66.6666%}.row .column.column-75{flex:0 0 75%;max-
→width:75%}.row .column.column-80{flex:0 0 80%;max-width:80%}.row .column.column-90
→{flex:0 0 90%;max-width:90%}.row .column .column-top{align-self:flex-start}.row .
→column .column-bottom{align-self:flex-end}.row .column .column-center{align-
→self:center}@media (min-width: 40rem){.row{flex-direction:row;margin-left:-1.0rem;
→width:calc(100% + 2.0rem)}.row .column{margin-bottom:inherit;padding:0 1.0rem}}a{color:
→#9b4dca;text-decoration:none}a:focus,a:hover{color:#606c76}dl,ol,ul{list-style:none;
→margin-top:0;padding-left:0}dl dl,dl ol,dl ul,ol dl,ol ol,ol ul,ul dl,ul ol,ul ul{font-
→size:90%;margin:1.5rem 0 1.5rem 3.0rem}ol{list-style:decimal inside}ul{list-
→style:circle inside}.button,button,dd,dt,li{margin-bottom:1.0rem}fieldset,input,select,
→textarea{margin-bottom:1.5rem}blockquote,dl,figure,form,ol,p,pre,table,ul{margin-
→bottom:2.5rem}table{border-spacing:0;display:block;overflow-x:auto;text-align:left;
→width:100%}td,th{border-bottom:0.1rem solid #e1e1e1;padding:1.2rem 1.5rem}td:first-
→child,th:first-child{padding-left:0}td:last-child,th:last-child{padding-right:0}@media
→(min-width: 40rem){table{display:table;overflow-x:initial}}b,strong{font-weight:bold}p
→{margin-top:0}h1,h2,h3,h4,h5,h6{font-weight:300;letter-spacing:-.1rem;margin-bottom:2.
→0rem;margin-top:0}h1{font-size:4.6rem;line-height:1.2}h2{font-size:3.6rem;line-
→height:1.25}h3{font-size:2.8rem;line-height:1.3}h4{font-size:2.2rem;letter-spacing:-.
→08rem;line-height:1.35}h5{font-size:1.8rem;letter-spacing:-.05rem;line-height:1.5}h6
→{font-size:1.6rem;letter-spacing:0;line-height:1.4}img{max-width:100%}.clearfix:after
→{clear:both;content:' ';display:table}.float-left{float:left}.float-right{float:right}
/* General */
html{font-size:55%}body{color:#000}.section-title blockquote{border:.3rem solid
→#d1d1d1;background-color:#90c0d8}h3{margin-bottom:.2rem}.container{margin:0;max-
→width:100%}hr{border-top:.3rem solid #d1d1d1;margin:2rem 0}ul{list-style:none}.
→location-details li{margin-bottom:0}@media print{.section-title:not(:first-child){page-
→break-before:always}.keepetogether{break-inside:avoid}}
/* Responsive tables */
.rtable{margin:0 0 40px 0;width:100%;box-shadow:0 1px 3px rgba(0,0,0,.2);
→display:table}@media screen and (max-width:580px){.rtable{display:block}}.rrow
→{display:table-row}.rrow:nth-of-type(odd){background-color:#fff}.rrow.rheader{font-
→weight:600;background:#d1d1d1}@media screen and (max-width:1024px){.rrow{padding:14px
→ 0 7px;display:block;border-bottom:1px solid #d1d1d1}.rrow.rheader{padding:0;height:6px}
→.rrow.rheader .rcell{display:none}.rrow .rcell{margin-bottom:10px;border:none}.rrow .
→rcell:before{margin-bottom:3px;content:attr(data-title);min-width:98px;font-size:.85em;
→line-height:10px;font-weight:700;text-transform:uppercase;display:block}}.rcell
→{padding:6px 12px;display:table-cell;border-bottom:1px solid #d1d1d1}.rcell ul{list-
→style:none}.rcell li{margin-bottom:0}@media screen and (max-width:1024px){.rcell
→{padding:2px 16px;display:block}}@media print{html{font-size:40%}.rtable{box-
→shadow:none;border:1px solid #d1d1d1}.rcell{border:1px solid #d1d1d1}.location-details
→.row{flex-direction:row}}

```

(continues on next page)

(continued from previous page)

```

</style></head>
<body>
  <header>
    <h1>Heritage Assets</h1>
  </header>
  <main>
    {% for resource in resources %}
    {% with resource_data=resource.resource %}
    <section class="section-title">
      <blockquote>
        {% if resource_data|has_key:"Heritage Asset Names" %}
        {% for n in resource_data|val_from_key:"Heritage Asset Names" %}
        {% if n|has_key:"Asset Name Use Type" %}
        {% if n|val_from_key:"Asset Name Use Type"|val_from_key:"@display_value"
↪ " == "Primary" %}
        <h2>{{ n|val_from_key:"Asset Name"|val_from_key:"@display_value" }}
↪ </h2>
        {% endif %}
        {% endif %}
        {% endfor %}
        {% endif %}
        <p><strong>Primary Reference Number: </strong>{{ resource_data|val_from_
↪ key:"System Reference Numbers"|val_from_key:"PrimaryReferenceNumber"|val_from_key:
↪ "Primary Reference Number"|val_from_key:"@display_value" }}<br>
↪ <strong>ResourceID: </strong>{{ resource_data.resourceinstanceid }}
        </p>
      </blockquote>
    </section>
    <section>
      <div class="container location-details">
        {% if resource_data|has_key:"Location Data" %}
        <div class="row">
          <div class="column">
            <div>
              <h3>OSGB Reference</h3>
              <p>{% if resource_data|val_from_key:"Location Data"|has_key:
↪ "National Grid References" %}
              {{ resource_data|val_from_key:"Location Data"|val_from_key:
↪ "National Grid References"|val_from_key:"National Grid Reference"|val_from_key:
↪ "@display_value" }}
              {% endif %}
              </p>
            </div>
          </div>
          <div class="column">
            <div class="keeptogether"></div>
            <div class="keeptogether">
              <div>
                <h3>Named Location</h3>
                <p>{% if resource_data|val_from_key:"Location Data"|has_key:
↪ "Addresses" %}
                {% for address in resource_data|val_from_key:"Location Data"|val_

```

(continues on next page)

(continued from previous page)

```

↪from_key:"Addresses"|val_from_key:"@display_value" %}
        {% if address|has_key:"Address Status" %}
        {% if address|val_from_key:"Address Status"|val_from_key:
↪"@display_value" == "Primary" %}
                {{ address|val_from_key:"Full Address"|val_from_key:
↪"@display_value" }}
        {% endif %}
        {% endif %}
    {% endfor %}
    {% endif %}
</p>
</div>
</div>
<div class="column">
    <div class="kepttogether"></div>
    <div class="kepttogether">
        <div>
            <h3>Localities/Administrative Areas</h3>
            <p>{% if resource_data|val_from_key:"Location Data"|has_key:
↪"Localities/Administrative Areas" %}
                {% for area in resource_data|val_from_key:"Location Data"|val_
↪from_key:"Localities/Administrative Areas" %}
                    <b>{{ area|val_from_key:"Area Type"|val_from_key:"@value" }}:
↪</b> {{ area|val_from_key:"Area Names"|val_from_key:"Area Name"|val_from_key:"@display_
↪value" }}
                {% endfor %}
            {% endif %}
            </p>
        </div>
    </div>
</div>
</div>
    {% endif %}
</div>
</section>
<hr>
<section>
    <div class="container">
        {% if resource_data|has_key:"Descriptions" %}
        <div class="kepttogether">
            {% for desc in resource_data|val_from_key:"Descriptions" %}
            <h3>{{ desc|val_from_key:"Description Type"|val_from_key:"@display_
↪value" }}</h3>
            <p>{{ desc|val_from_key:"Description"|val_from_key:"@display_value" }}
↪</p>
            {% endfor %}
        </div>
        {% endif %}
        {% if resource_data|has_key:"External Cross References" %}
        <div class="kepttogether">
            <h3>External Cross References</h3>

```

(continues on next page)

(continued from previous page)

```

<div class="rtable">
  <div class="rrow rheader">
    <div class="rcell">Number</div>
    <div class="rcell">Description</div>
    <div class="rcell">Source</div>
  </div>
  {% for src in resource_data|val_from_key:"External Cross References
↪ " %}
    <div class="rrow">
      <div class="rcell" data-title="Number">{{ src|val_from_key:
↪ "External Cross Reference Number"|val_from_key:"@display_value" }}</div>
      <div class="rcell" data-title="Description">
        {% if src|has_key:"External Cross Reference Notes" %}
        {{ src|val_from_key:"External Cross Reference Notes"|val_from_
↪ key:"External Cross Reference Description"|val_from_key:"@display_value" }}
        {% endif %}
      </div>
      <div class="rcell" data-title="Source">{{ src|val_from_key:
↪ "External Cross Reference Source"|val_from_key:"@display_value" }}</div>
    </div>
  {% endfor %}
</div>
</div>
{% endif %}
</div>
</section>
{% endwith%}
{% endfor %}
</main>
</body>
</html>

```

Customizing HTML Email Templates

In addition to the standard email templates that are provided by Arches, from version 7.5 is possible to customize (“customise”, if not spelled in the US) HTML email templates in your Arches project to include such things as branding, logos, custom text and other elements specific to a given instance.

A developer will need to customize the starting templates and add any extra context items to the settings file which are required by the templates.

Templates and their Locations

Main Templates

The starting templates provided when you create a new project are:

- Download Read Email Notification
- General Notification
- Package Load Completion Notification

These templates are located in the following directory: *<project name>/templates/emails/*

They overwrite the out-of-the-box templates found in the arches install directory. If you do not wish the arches install directory versions to be overwritten then remove these templates from the project directory.

They consist of static text and variables. The variables are used to render the email with the appropriate context items. The variables are enclosed in double curly braces, e.g. `{{ variable_name }}`.

Templates for Common Styling and Formatting

There are also three files supplied which are referenced in the above templates as includes. These files allow for common styling and formatting to be applied to all the HTML email templates. These files are: - `custom_email_css.htm` - `custom_email_footer.htm` - `custom_email_header.htm`

Initially the above files are empty.

The files are located in the following directory: *<project name>/templates*

When adding an image logo, you may wish to do so in Base64 encoded format to ensure the images appear as expected in the email. There are a number of online tools that can be used to convert an image to Base64 format.

Extra Context Items

In some instances, you may wish to add extra context items which are used by the template to render the email. These context items are stored in the `EXTRA_EMAIL_CONTEXT` setting within the `settings.py` or `settings_local.py` file.

The starting default `EXTRA_EMAIL_CONTEXT` object contains the value for Salutation and a expiration date that is based on the `CELERY_SEARCH_EXPORT_EXPIRES` setting.

For the sake of consistency, if you are using common templated text across templates while not in the default context items, it is recommended that you add it to the `EXTRA_EMAIL_CONTEXT` setting.

For example, if you have a common email address that you wish to use across all templates, you could add it to the `EXTRA_EMAIL_CONTEXT` setting as follows:

```
EXTRA_EMAIL_CONTEXT = {
    "salutation": ("Hi"),
    "expiration":(
        datetime.now()
        + timedelta(seconds=CELERY_SEARCH_EXPORT_EXPIRES)
    ).strftime("%A, %d %B %Y"),
    "email_address": ("person@example.org"),
}
```

You would then be able to use the tag `{{ email_address }}` to render the email address in your template(s).

Other Considerations

You are advised, when creating customized HTML email templates, to ensure the templates are accessible. For further information on configuring your templates to be accessible, see the [Accessibility](#) section.

Accessibility

As of version 7.5, Arches can be configured to meet WCAG defined AA level accessibility requirements for all public-facing user interfaces. Please review the documentation on activating the Arches [Accessibility Mode](#).

Please continue reading below to understand how to better meet accessibility requirements for your customization of Arches.

Contents

- [Summary](#)
- [Tools Used](#)
- [Key Points](#)
 - [Color Contrast](#)
 - [Form Fields](#)
 - [Headings](#)
 - [Links](#)
 - [Keyboard](#)
 - [Responsive Design](#)
 - [HTML Validation](#)
 - [Screen Reader](#)
- [Alternative solutions where components cannot be made accessible](#)
- [Additional Points](#)

Summary

It is important that Arches is developed with inclusivity in mind by making it accessible to users with disabilities.

In a number of regions, organisations are required to ensure that any software they use, or provide as a service, is accessible for users with disabilities. To this end, any UI development within Arches must take measures to conform to the guidance set out in the WCAG 2.1 requirements. This will allow Arches to be more easily adopted by such organisations and provide benefits to a wider audience.

The following information details the minimum steps required to adhere to WCAG accessibility guidelines. Although the remit has been to adhere to AA standards, wherever possible AAA has been reached for issues such as color contrast.

Here's a link to all the [WCAG 2.1 requirements](#)

You should ensure that you have developed and tested any code against these as a minimum when submitting code back to the project.

Tools Used

- Browsers - Chrome, Firefox and Safari (on iOS via Browserstack)
- Browser extensions (all free, no sign-up required apart from Browserstack):
 - [Wave](#)
 - [Lighthouse](#)
 - [Browserstack](#) - requires paid subscription
 - [Landmarks](#)
 - [NVDA](#)
 - [W3C HTML Validator](#)
 - [Contrast checker](#)

Key Points

Although many files have been worked on for the many different requirements, there have been some frequently identified issues. Here are the commonly found problems:

Color Contrast

Note: Tools used: [Wave](#) / [Contrast checker](#)

Using a combination of the Wave browser extension and the Contrast Checker website mentioned above, you can identify what elements on a page that need changing, for example, from the Arches v5 demo site, take the “Resource Type” button on the search page:

It has a background color #579DDB and a foreground color #FFFFFF - this fails the contrast test. You can use the contrast checker to test how things look when you lighten or darken either the background or foreground. In this instance, using the slider, let’s darken the background color to be #1E5A8F instead, which passes WCAG AAA.

Form Fields

Note: Tools used: [Wave](#) / [Lighthouse](#) / [W3C HTML Validator](#) / [NVDA](#)

Ensure form fields have correct labelling.

For example, instead of:

```
<h5>Field label text</h5>
<input type="text">
```

use this code instead:

```
<label for="myField">Field label text</label>
<input type="text" id="myField">
```

Sometimes it may suit design purposes not to have a label and make use of placeholder text. This is fine, but be mindful that users using screen readers will not get placeholder text read out to them. So we can make use of the `aria-label` attribute:

```
<input type="text" id="myField" placeholder="Field label text" aria-label=
↪ "Field label text">
```

...or using `aria-labelledby`:

```
<span id="someText">Field label text</span>
<input type="text" aria-labelledby="someText">
```

Also, you can use the `aria-label` attribute on a container element to describe the content within:

```
<div class="container" aria-label="Search buttons to filter the search results
↪ ">
  <button id="filterBtn">Filters</button>
  <button id="typeBtn">Type</button>
</div>
```

Headings

Note: Tools used: [Wave](#) / [Landmarks](#)

Make sure that all headings are ordered and nested correctly. There should only be one `<h1>` tag per page, and be sure to not skip any heading levels. The correct order should be something like this:

```
<h1>Main Heading</h1>
<h2>Navigation Menu</h2>
<h2>Sidebar</h2>
  <h3>Profile</h3>
  <h3>Settings</h3>
  <h3>Help</h3>
  ...
```

Links

Note: Tools used: [Wave](#) / [W3C HTML Validator](#)

If a link contains no text, then the function or purpose will not be understood by screen reader users.

For example:

```
<a href="">View more...</a>
or
<a>View more...</a>
```

...should be:

```
<a href="#" aria-label="View more search results">View more...</a>
```

...note the use of an `aria-label` to provide a clearer description of what the link is for.

Keyboard

Note: Tools used: none (manual checks required)

UI development must ensure the website/page is still navigable and actionable via the keyboard. There may be instances where click events are required on elements other than `href` links, for example (using Knockout binding):

```
<div class="css-class" data-bind="click: function() {myFunc();}">
  Some content
</div>
```

This will listen for a mouse click on the `div` element, but this won't work if a user is using their keyboard to navigate and operate the website. A keyboard user will not be able to `tab` to this element or be able to action it by pressing their space bar or enter key. To facilitate this, we need to make it `tabbable` and actionable via a `keypress` as follows:

```
<div class="css-class"
  tabindex="0"
  data-bind="click: function() {myFunc();}"
  onkeypress="$(this).trigger('click');">
  Some content
</div>
```

Note the use of `tabindex="0"` which includes the element within the natural DOM tab order and the `onkeypress` which in this example uses jQuery to force a `click`. There may be several ways to achieve this but always ensure any clickable element can also be actioned using a keyboard, usually the enter key once tabbed to.

Responsive Design

Note: Tools used: [Lighthouse](#) / [Browserstack](#) / Browsers (Chrome, Firefox and Safari)

When designing websites, we must think about all users and not for example, only desktop or laptop users with large screens. Users with visual impairment may increase the font size or spacing, or possibly the screen resolution may be lower.

By developing a responsive application, users making these adjustments will benefit from the application adjusting correctly to it. The application will also benefit from this by being available on tablets and mobile devices and in some regions, mobile phones are peoples' only computing device.

The website should offer the same functionality whether viewing on a large monitor or mobile screen and anything in between so that we can be as inclusive as possible. If certain information cannot be viewed on a smaller screens, then a suitable alternative should be presented to the user.

Arches uses the javascript library called Bootstrap which enables the content to be rendered in a grid system that can be adapted to suit varying screen sizes and types, including mobiles and tablets. No content should appear 'cut-off' when reducing the screen width; it should either stack, wrap or be presented differently.

This can easily be tested in a browser such as Chrome or Firefox which have built in developer tools for viewing at different devices or screen widths. Of course the ultimate test would be to use an actual device to see what happens in the real world. For this level of testing I would recommend Browserstack which has access to many different physical devices and browsers.

It's also good practice to ensure that web pages operate the same using different web browsers. For example, some things may not work correctly in Safari or Chrome, but everything seems fine in Firefox.

HTML Validation

Note: Tools used: [W3C HTML Validator](#)

Any rendered html needs to pass [W3C HTML Validator](#) tests. With any dynamically produced web page, it's easy to load the page in a browser and view the source, copy and paste into the 'Validate by direct input' form field, run the test and work on any errors as necessary.

Here are some common issues found:

- Empty id and class attributes, like `id=""` and `class=""` - if they're empty remove them
- Incorrect html markup, like having a div tag inside a span tag
- Incorrect html5 semantic markup - for example no landmarks, no header, no main, no footer etc
- On some pages, the first code on a page contains the open source copyright comment, which is acceptable and required by the GNU Affero General Public License, but sometimes the comment is duplicated causing a validation error

Screen Reader

Always be mindful of users that require to use screen readers and check how sections of the page are read out and in what order.

For desktop checks, use the [NVDA](#) application to identify possible changes or where to include some `aria-label` descriptive text to assist with the content visualisation.

Mobile devices have some built in screen reader technology. For iOS it's called **Voice Over** and can be accessed under **Settings>Accessibility**. For Android devices it's called **Screen Reader** and can be accessed via **Settings>Accessibility>Screen reader**.

For example, when viewing a web page, one of the first things read out may be the menu. If the menu has many items, this could become a tedious activity, so it's good practice to include a "Skip to main content" link that appears when a user first presses the tab button. Pressing enter should change focus to the start of the main content, bypassing the menu items.

Alternative solutions where components cannot be made accessible

In the event that a specific component cannot be made fully accessible, an alternative method of achieving the same outcome should be provided.

For example, if using an SVG canvas type library to display information or provide a search function, a tabular alternative could also be created that provides the same function.

Ideally, the accessible solution would be the primary solution.

Additional Points

There are many more WCAG guidelines that need to be adhered to but these mentioned here are among the most common. It's always good practice to have these points in mind whenever creating web pages/content. Always keep in mind how a keyboard-only user would be able to interact with pages and how they would still work on smaller devices such as tablets or mobiles.

Even though your targeted users may not be using mobile devices, you have to cater for every need. In this day and age, the “**Mobile first**” principle should be used and play a significant role in any product design/development work.

Integrating Arches with ArcGIS

<https://github.com/archesproject/arches-docs/issues/216>

Localizing Arches

If you want to support localization in your Arches instance, you'll first need to do the following:

1. Update your settings.py file by adding this import statement at the top:

```
from django.utils.translation import gettext_lazy as _
```

2. Next copy the MIDDLEWARE setting to your project's settings.py file. If it's already in your settings.py file, be sure to uncomment `"django.middleware.locale.LocaleMiddleware"`

```
MIDDLEWARE = [  
    # 'debug_toolbar.middleware.DebugToolbarMiddleware',  
    "corsheaders.middleware.CorsMiddleware",  
    "django.middleware.security.SecurityMiddleware",  
    "django.contrib.sessions.middleware.SessionMiddleware",  
    #'arches.app.utils.middleware.TokenMiddleware',  
    "django.middleware.locale.LocaleMiddleware",  
    "django.middleware.common.CommonMiddleware",  
    "django.middleware.csrf.CsrfViewMiddleware",  
    "arches.app.utils.middleware.ModifyAuthorizationHeader",  
    "oauth2_provider.middleware.OAuth2TokenMiddleware",  
    "django.contrib.auth.middleware.AuthenticationMiddleware",  
    "django.contrib.messages.middleware.MessageMiddleware",  
    # "django.middleware.clickjacking.XFrameOptionsMiddleware",  
    "arches.app.utils.middleware.SetAnonymousUser",  
]
```

- Next add the LANGUAGE_CODE, LANGUAGES, and SHOW_LANGUAGE_SWITCH to your project's settings.py file and update them to reflect your project's requirements:

```
# default language of the application
# language code needs to be all lower case with the form:
# {langcode}-{regioncode} eg: en, en-gb ....
# a list of language codes can be found here http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = "en"
# list of languages to display in the language switcher,
# if left empty or with a single entry then the switch won't be displayed
# language codes need to be all lower case with the form:
# {langcode}-{regioncode} eg: en, en-gb ....
# a list of language codes can be found here http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGES = [
    ('de', ('German')),
    ('en', ('English')),
    ('en-gb', ('British English')),
    ('es', ('Spanish')),
]
# override this to permanently display/hide the language switcher
SHOW_LANGUAGE_SWITCH = len(LANGUAGES) > 1
```

- Now add this import statement to the top of your urls.py file:

```
from django.conf.urls.i18n import i18n_patterns
```

- Finally add the following code to the end of your urls.py file:

```
if settings.SHOW_LANGUAGE_SWITCH is True:
    urlpatterns = i18n_patterns(*urlpatterns)
```

Once the system is prepared for localization, the next steps involve generating a Django message file or .po file which will contain all available translation strings in Arches and how they should be translated in any given language.

For more information, see [Localization: how to create language files](#) in the Django documentation.

There are some example commands to make and load PO files in the core arches settings file that can be found [here](#). If loading a new PO file, simply replace the existing po file and run compilemessages.

Localizing Graph Strings within Arches

You can also export strings from your arches graphs for localization using the following arches-specific command

```
python manage.py i18n makemessages
```

You can import them with the following command

```
python manage.py i18n loadmessages
```

This will attempt to load the graph translation files (graph.po files) for every language specified in the LANGUAGES array from settings.py.

Setting up Localized Languages for Business Data

By default, every language from the `LANGUAGES` array in `settings.py` is available for business data entry. To add additional languages for business data entry only, you can do the following.

1. Access the admin page (<http://localhost:8000/admin/>)
2. Choose the “Languages” table. (<http://localhost:8000/models/language>)
3. Select “Add Language”
4. Fill in information on new language, including a default direction.

Repeat this process for all new languages you wish to add.

Additionally, remove any languages you do not plan on using.

Once this is complete, text widgets should be able to write data in the desired languages.

RDF Imports and Exports

Business data can be exported in RDF format. The directionality of the string data will be lost as the RDF specification does not include directionality. There is an [active attempt](#) to include direction within the RDF specification.

CSV Exports and Imports

It is possible to import and export localized business data through CSV format. There is a `--language` switch that will limit the languages that will be exported (all languages are exported by default). However, if attempting to re-import a limited subset of languages through the csv importer, entire string objects will be overwritten by the subset. For example, if a string node has values for English, Spanish, and French, the subset of languages can be limited by specifying

```
--languages en,es
```

If attempting to import the resulting csv, any values that were pre-existing for French would be overwritten in “overwrite” mode or added as a separate tile in “append” mode. There is currently no way to merge these values. If the intention is to re-import the csv values later, export all languages.

Managing and Hosting IIIF Servers

Arches is configured to use Cantaloupe if you want to host images made available via the IIIF presentation API. Below is a simplified setup guide. The full Cantaloupe setup documentation is [here](#)

Setting Up Cantaloupe

1. Download and extract/unzip the cantaloupe source code from among [these releases](#) . We recommend the latest release of version 4.
2. In a directory containing all the contents of the downloaded source code, make a copy of `cantaloupe.properties.sample` and name it `cantaloupe.properties`. When hosting images locally (relative to your arches project), change the value for argument: `FilesystemSource.BasicLookupStrategy.path_prefix` to the absolute path of wherever your uploaded files are located, for example `/home/ubuntu/project/project/uploadedfiles/`.

“Lookup Strategy” should already be set to “BasicLookupStrategy”.

Note: Other strategies (such as delegation) can be configured depending on your desired implementation.

3. Ensure that the argument CANTALOUPE_DIR in your project’s settings.py file is os.path.join(APP_ROOT, "uploadedfiles") if your project’s uploadedfiles directory is where images will be stored, otherwise point to the appropriate location.
4. Run the Cantaloupe server (either using the java command or some service or process manager; see the “[Running](#)” section of Cantaloupe docs)

Note: Remote hosting of Cantaloupe server, the manifest.json files, and image files are all still in development.

Creating IIIF Manifests / Image Services

The IIIF Manifests each represent a collection of at least one image (called a “canvas”). It is called an Image “Service” because the cantaloupe server enables the user to zoom and dynamically view the image.

Navigate to the Image Service Manager in the Arches UI and select at least one image to create a new service. If you do not see the icon for Image Service Manager in the left-hand navbar, you may need to update the entry in the Plugins table of your database like so:

```
sudo -u postgres psql -d [test_project] -c "update plugins set config = '{"show":true}'
↳where name = 'Image Service Manager';"
```

Now that an Image Service (referred to as a “Manifest”) exists, it will be available for any user to create Annotation data. You can edit this Image Service in the Image Service Manager to upload additional image files or add metadata. When a resource is edited and a tile saved to that card on that model, if the file is an image type (i.e. a .tiff, .tif, .jpg, .jpeg, or .png) a record in the iiif_manifests table in the database will be created pointing to a manifest .json file that will render the image file from cantaloupe into the IIIF Viewer card (see below).

IIIF Viewer / Annotation data

1. To make use of IIIF imagery, a resource model must have a semantic node configured to use the “IIIF Card” selected for “Card Type”.
2. Inside this card/nodegroup, add a child node and select “annotation” datatype. To include other data along with this annotation, (e.g. text, date, or related resources) create sibling nodes of those datatypes, ensuring they are still the children of the semantic node designated with the “IIIF Card”.
3. When creating a tile for this card in the resource editor, the user will first be prompted to select a IIIF Manifest from a dropdown list. You should see any IIIF Manifests created from the above process.

Note: A single tile for a IIIF card could contain multiple features (point, line, polygon) as part of the annotation data, but commonly you would also want nodes of other datatypes (for ex: string) grouped into this IIIF card; thus to make multiple tiles with different values on the same resource instance, you need to check “Allow Multiple Values” on the IIIF card in the Card Manager.

Populating IIIF Manifest Dropdown Lists

A dropdown list provides users with options for selecting between IIIF manifests when they use the resource editor. A user can also add a new IIIF manifest that exists on a remote server by pasting the URL to that manifest (the URL will point directly to the remote server's manifest JSON resource) into the input/search box of the dropdown list. See the animation below for an illustration:

One can use SQL to pre-populate the list of IIIF Manifests. The following SQL inserts will pre-populate the IIIF manifest dropdown list:

```
insert into iiif_manifests (label, url, description) values ('IIIF Manifest of Gospel_
↳Book', 'https://media.getty.edu/iiif/manifest/a628a212-a325-406c-aa4d-c43eeb393ec5',
↳'accession number: 83.MB.69, TMS ID: 1571, UUID: 8c6116d5-09f6-4416-8d15-1804c9337c65
↳');

insert into iiif_manifests (label, url, description) values ('IIIF Manifest of Saint_
↳Matthew Seated', 'https://media.getty.edu/iiif/manifest/028b269e-054f-4d39-83b9-
↳6b207707731d','accession number: 83.MB.69.9v, TMS ID: 3275, UUID: 4093369e-678b-41fc-
↳a7e9-a5fef60c7385');

insert into iiif_manifests (label, url, description) values ('IIIF Manifest of The_
↳Transfiguration', 'https://media.getty.edu/iiif/manifest/a91a88a3-ca07-480f-b749-
↳8e1c28d4f040','TMS ID: 3278, UUID: 601d907b-2941-4724-9f14-7b7d22f2be63');
```

More information:

- General information on using IIIF (Cantaloupe version 3 only, but still useful): <https://iiif.github.io/training/intro-to-iiif/>
- Cantaloupe Documentation: <https://cantaloupe-project.github.io/manual/4.1/getting-started.html>
- IIIF Presentation API Documentation: <https://iiif.io/api/presentation/2.1/>
- IIIF Image API Documentation: <https://iiif.io/api/image/2.1/>

Task Management

Dependencies

Task management using Celery is available in Arches if you have a message broker like RabbitMQ or Redis. The Celery documentation provides information on [broker installation](#).

Configuration

Once you have your broker available, you will need to configure your settings. You will find this in your project's settings.py file. Each setting that begins with the CELERY prefix will be used as a celery config, so you can configure celery by adding the configs you need

```
CELERY_BROKER_URL = 'amqp://guest:guest@localhost'
CELERY_RESULT_BACKEND = 'django-db' # Use 'django-cache' if you want to use your cache_
↳as your backend
```

The settings you are likely to want to modify right away are the CELERY_BROKER_URL and the CELERY_RESULT_BACKEND. Your CELERY_BROKER_URL should point to your broker's service URL. If you are

using RabbitMQ, you will probably want to create a new user and password and replace ‘guest:guest’ with the new users credentials.

Your CELERY_RESULT_BACKEND is set to the Django ORM by default. If your task will be run frequently, you may want to consider a more performant option. If you’ve configured a cache for django, you could use that, or you could use another [backend option](#) that Celery supports.

Adding Tasks to Your Project

To add additional tasks you your project, all you need to do is add a tasks.py file to your project. This could be placed in your project’s root directory (next to manage.py) or any sub-directory. However, you will likely want to put it - at least to start in the directory below root (next to urls.py).

Here’s an example of a very simple task in tasks.py:

```
from __future__ import absolute_import, unicode_literals
from celery import shared_task

@shared_task
def add(x, y):
    return x + y
```

To call your task, just import tasks into the module that will run it.

Running Celery

For your tasks to run both your broker service and a Celery worker need to be running. For production you will likely want to run a worker as service. One way to do this is to use supervisord. For more information see: [Setting up Supervisord for Celery](#)

However for development, you probably want to run your worker in a terminal. To do so just cd into your project’s root directory with your virtual environment activated and run:

```
python manage.py celery start
```

Users of Apple Silicon Macs may encounter `billiard.exceptions.WorkerLostError` following a warning emitted by the OS explaining it is “crashing instead” rather than unsafely calling `fork()`. Until this [issue](#) is resolved by celery, launching like so will silence the errors:

```
OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES python manage.py celery start
```

Note that in the event of celery errors which do not clearly indicate what is breaking or preventing your tasks from succeeding, we recommend running the following command instead for the purpose of debugging and isolating any unhandled exceptions:

```
celery -A [app_name] worker --loglevel=warning
```

Once the worker is running you should be able check your database and see you task result output with the following query:

```
select * from django_celery_results_taskresult;
```

If you want to monitor your tasks with a realtime console, you can use [Flower](#).

Two-factor Authentication

Two-factor authentication is an extra layer of security designed to ensure that you're the only person who can access your Arches account, even if someone knows your password.

- *How Two-factor Authentication Works*

A brief introduction to two-factor authentication theory.

- *Enabling Two-factor Authentication in Arches*

The specific mechanisms Arches uses for two-factor authentication, and how to enable two-factor authentication for your Arches application.

- *Setting up Two-factor Authentication for User Accounts*

How to set up two-factor authentication as an Arches user.

How Two-factor Authentication Works

Two-factor Authentication is the technical term for the process of requiring a user to verify their identity in two unique ways before they are granted access to the system. Users typically rely on authentication systems that require them to provide a unique identifier such as an email address or username and a correct password to gain access to the system. Two-factor Authentication extends this by adding an additional step that requires the user to enter a one-time dynamically generated token that has been delivered through a secondary method that presumably only the user has access to. This token is randomly generated and lasts a brief period of time before changing. It is based on an encrypted secret key that is stored in the application and secondary system (eg. smartphone).

Two-factor Authentication gives the user and system administrator a peace of mind that even if the user's password is compromised, the account cannot be accessed without also knowing the dynamically generated one-time password.

Enabling Two-factor Authentication in Arches

There are two configurable settings, `ENABLE_TWO_FACTOR_AUTHENTICATION` and `FORCE_TWO_FACTOR_AUTHENTICATION`. Each accepts a value of True or False.

- `ENABLE_TWO_FACTOR_AUTHENTICATION` - Allows users to enable two-factor authentication via their `UserProfile`, and redirects login of users that have enabled two-factor authentication to secondary credentials page.
- `FORCE_TWO_FACTOR_AUTHENTICATION` - Must have `ENABLE_TWO_FACTOR_AUTHENTICATION` enabled. Forces all users to log in with two-factor authentication credentials.

Note: `ENABLE_TWO_FACTOR_AUTHENTICATION` and `FORCE_TWO_FACTOR_AUTHENTICATION` do not trigger any other actions, such as terminating user sessions.

Setting up Two-factor Authentication for User Accounts

If `ENABLE_TWO_FACTOR_AUTHENTICATION` or `FORCE_TWO_FACTOR_AUTHENTICATION` have been enabled in your Arches application, users can check the status of their accounts in the User Profile page.

The screenshot shows the 'Profile Manager' interface. At the top, it says 'Welcome, admin'. Below this, a dark blue bar displays 'User name: admin'. The main content area is divided into two columns. The left column is labeled 'Account' and lists fields: 'admin' (User name), '*****' (Contact email), 'None' (Phone), and 'DISABLED' (Two-Factor Authentication). The right column has an 'Edit' button and a 'Change password' link. The 'Two-Factor Authentication' status is highlighted in red.

Fig. 39: User Profile showing two-factor authentication status.

From User Profile Edit page, Users can send an email to their registered email address containing instructions and a link to set up two-factor authentication.

The screenshot shows the 'Send Two-Factor Authentication email?' dialog box. The dialog box has a blue header with the title and a question: 'This will send an email containing instructions to change two-factor authentication settings to your registered email address. Are you sure you would like to proceed?'. There are 'Cancel' and 'OK' buttons. Below the dialog box, the 'User name: admin' is displayed. The main content area is divided into two columns. The left column is labeled 'Account' and lists fields: 'admin' (Arches user name), 'First name', 'Last name', 'Phone Number (optional)', 'Contact Email', and 'Two-Factor Authentication'. The right column has a 'Save' button, a 'Cancel' button, and a 'Click here to update Two-Factor Authentication settings via email' link. The 'Two-Factor Authentication' status is highlighted in red.

Fig. 40: User Profile showing two-factor authentication reset email interaction.

Note: In order to continue, the User should already have access to a means of secondary authentication. This is done with an external application, usually with [Google Authenticator](#), [Authy](#), [LastPass Authenticator](#), or any other

authentication application.

Following the email link, the user will navigate to the two-factor authentication settings page.

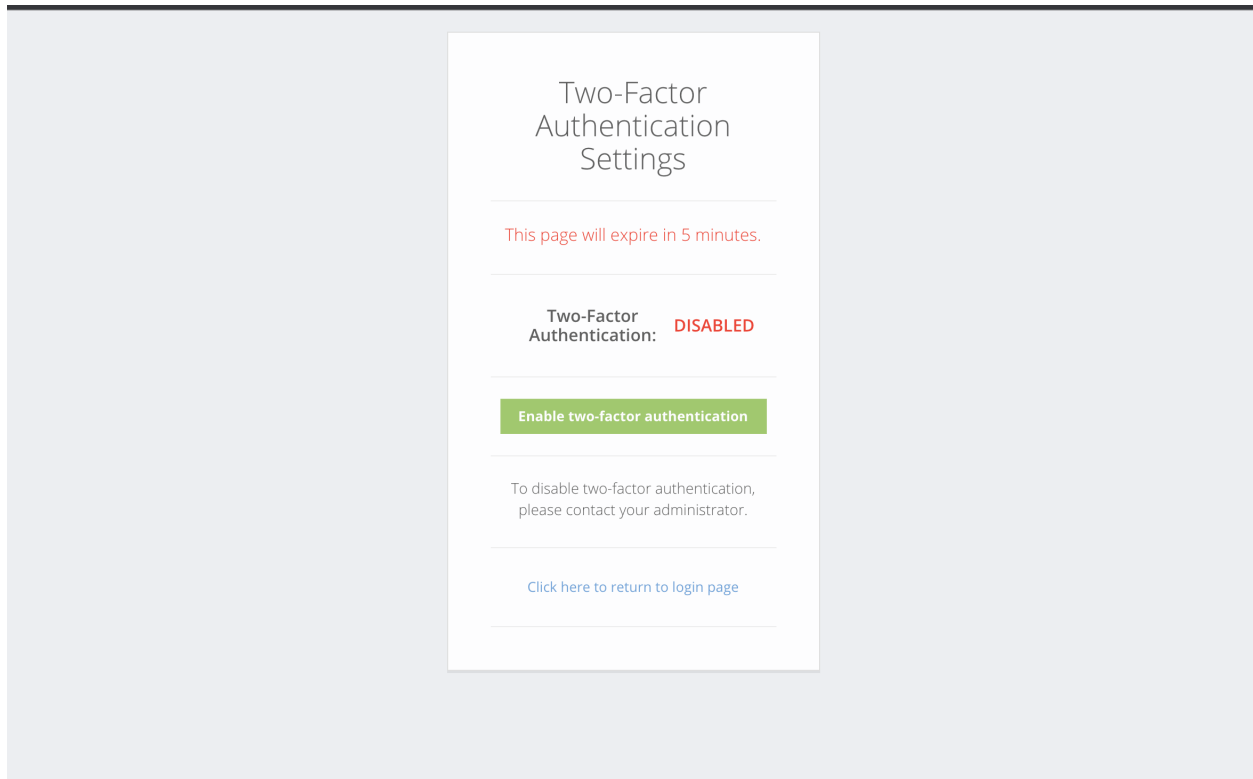


Fig. 41: Two-factor authentication settings page.

From this page, Users can generate a QR code to be scanned with an external authentication application, or a secret key to be entered manually. This secret is used to generate time-based authentication tokens.

Once the user has enabled two-factor authentication, or if `FORCE_TWO_FACTOR_AUTHENTICATION` has been enabled at the system level, the user will be presented with an additional step in the login process. Once the six-digit authentication code has been entered, the User will be logged in.

Using Arches Offline

<https://github.com/archesproject/arches-docs/issues/12>

Migrating Data from v3

Terminology Note

In v3 we had “resource graphs”, while in v4 and later we call these “Resource Models”. Conceptually they are the same. We’ll be referring to them here as “v3 graphs” and “Resource Models”, respectively.

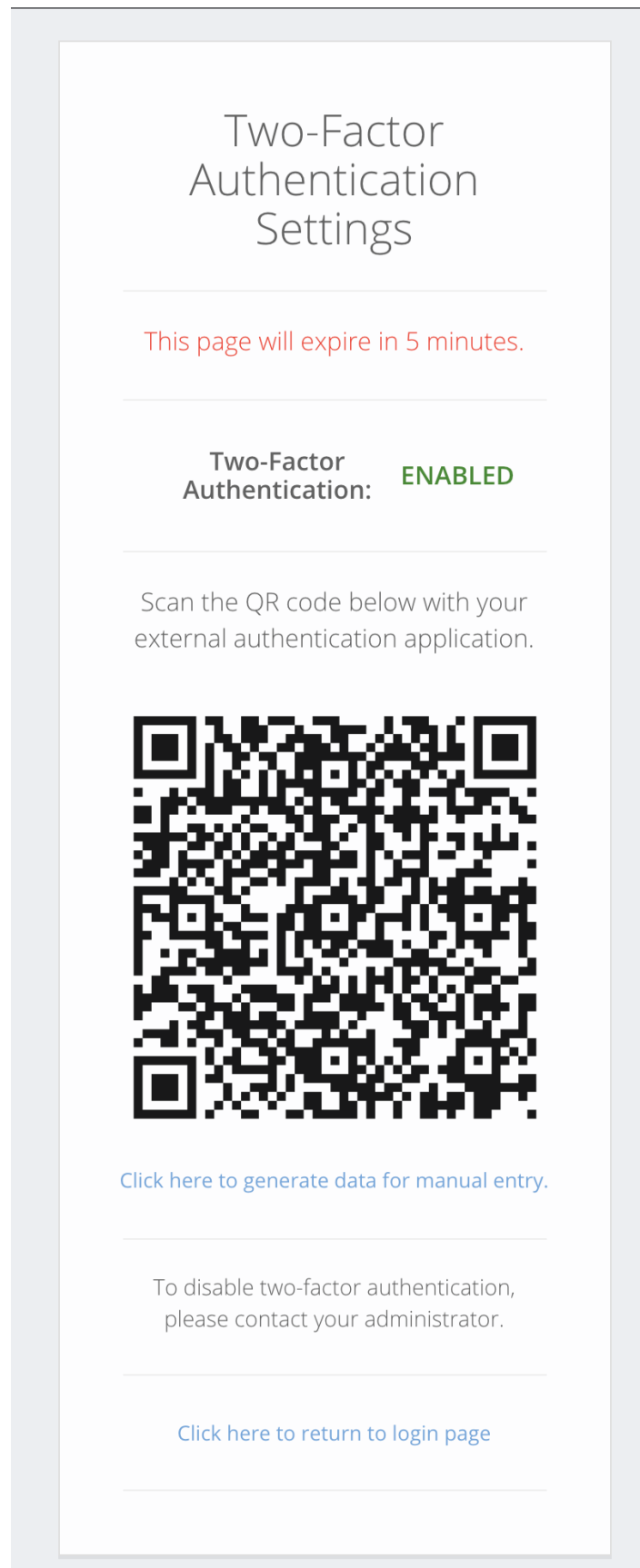
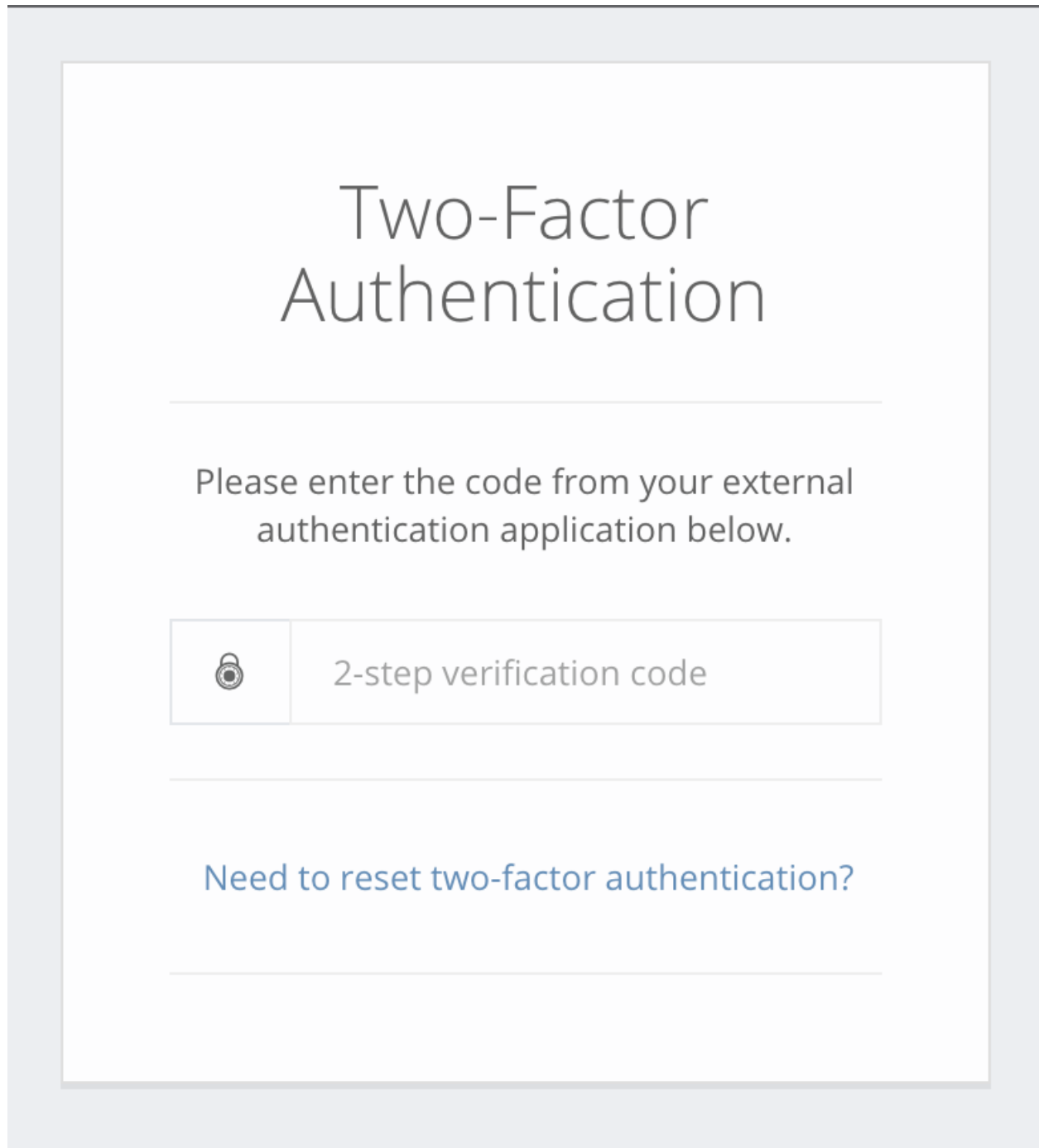



Fig. 42: Two-factor authentication settings page showing QR code.



The image shows a web page for Two-Factor Authentication. It has a light gray background with a white central box. The title 'Two-Factor Authentication' is centered at the top in a large, dark gray font. Below the title is a horizontal line. The text 'Please enter the code from your external authentication application below.' is centered in a dark gray font. Below this text is a form field with a light gray border. The form field is divided into two parts: a small square on the left containing a lock icon, and a larger rectangular area on the right containing the text '2-step verification code'. Below the form field is another horizontal line. At the bottom of the form area, the text 'Need to reset two-factor authentication?' is centered in a blue font. Below this text is a final horizontal line.

Two-Factor Authentication

Please enter the code from your external authentication application below.

 2-step verification code

[Need to reset two-factor authentication?](#)

Fig. 43: Two-factor authentication login page.

Note: In the following guides, you’ll see mention of “v4”. However, all of these steps work for Arches v5, as well.

Upgrading your Arches installation is a complex process, as a significant backend redesign was implemented in v4. We have developed the following documentation (and the code to support it) to guide you through the process. You will be performing a combination of shell commands and basic file manipulation.

Before migrating data, you’ll need to *install core Arches* and *create a new project*. You can name your project whatever you want, but throughout this documentation we’ll refer to it as `my_project`. You can customize the templates and images in your project any time (before or after migrating the data). We recommend adding a Mapbox key right away so you can use the map for visual checks during the migration.

See also:

Please see the main *installation guide*.

Before moving on, you must be able to run the Django development and view your project in a browser at `http://localhost:8000`.

Once you are ready, you can begin the migration process. The overall form of the process goes like this:

- Export data from existing v3 Arches installation
- Create a package (Arches-HIP users: this is already done for you)
- Place the v3 data into the Package directories
- Run commands to convert the v3 data to v4 data
- Load the package into any v4 project

Exporting Your Data From v3

You must export all of your data from v3. Before you begin, however, you’ll need to install some enhanced commands into your v3 app. This is a simple process:

1. Download and unzip [arches3-export-utils-master.zip](#) (source)
2. Copy the “management” directory into your v3 app alongside the `settings.py` file.
3. In your v3 environment, run `python manage.py v3v4 --help` to make sure the new commands have been installed.

Warning: Be sure to backup your v3 database before beginning the export process.

Now you are ready to begin exporting your data from v3. Follow these steps:

Export v3 Business Data

In your v3 command line run:

```
python manage.py v3v4 -o export-resources --format JSON
```

You will get a console update during the process, which could take a few minutes. The result will be one file:

- v3resources-all-<date>.json

Place the file(s) somewhere easy to access.

Important: If you have a very large database (maybe 25k+ resources), we recommend using `--format JSONL`. This will create a [JSON Lines](#) file, which requires minimal memory resources. Exporting the entire database to a single JSON file can crash servers without enough memory. For even more control over the export, add `--split` to the command above. One JSON/JSONL file will be created **per resource type**. This is extremely helpful for debugging migration issues.

Export v3 Resource Relations

In your v3 command line run:

```
python manage.py v3v4 -o export-relations
```

You will get a console update during the process, which could take a few minutes. The result will be one file:

- v3relations-all-<date>.csv

Place the file somewhere easy to access.

Export v3 Reference Data

In your v3 command line run:

```
python manage.py v3v4 -o export-skos --name Arches
```

The result will be one file:

- v3scheme-arches-<date>.xml

Place the file somewhere easy to access. This is the “Arches” scheme from your RDM, which is, typically, where your entire concept set will exist. If you are using a different concept scheme, substitute its name for “Arches” in the command above.

Warning: You are only able to migrate one scheme. If your v3 dropdown lists are composed of concepts from two different schemes (i.e. you added another scheme alongside “Arches”, added concepts to it, and then added those concepts to dropdown lists) you’ll need to manually consolidate these schemes into one before exporting.

Dropdown Lists themselves are not migrated, they are recreated in v4 based on Top Concepts.

Transfer all v3 uploaded media files

You must move all of the media files that have been uploaded to your v3 deployment to your v4 project.

By default, the directory in your new v4 project should be called `my_project/my_project/uploadedfiles`. If this directory doesn't exist, create it, and move all of the v3 media into it.

AWS S3 and Azure Users

You should be able to continue using the same storage bucket, and just point your v4 project at it. Just make sure your content is in a folder called `uploadedfiles`. **In theory this should work, but we haven't tested it.**

Now that you have exported all of the data you need from your v3 deployment, head back to [Migrating Your Data](#).

After you have all the v3 data exported, you are ready to follow the appropriate workflow for your deployment.

Migrating Your Data

The workflow you must use for the migration depends on the nature of your v3 deployment.

Arches-HIP App

If your v3 deployment of Arches was based on **Arches-HIP**, and you did not modify any of the graphs (beyond perhaps changing node names) you can use the Arches-HIP Workflow. If you have changed the RDM content that's fine, it will be preserved through the migration.

Arches-HIP Workflow

1. Download the prepared v4 HIP package.

Download [arches-v4-hip-pkg-master.zip](#) (source).

Unzip this directory and place it in your project. The result should look like this:

```
my_project/
├─ manage.py
├─ my_project/
├─ arches-v4-hip-pkg-master/
│   ├─ business_data/
│   ├─ extensions/
│   └─ (etc.)
```

If you want, you can rename the directory. For this tutorial, we will rename it from `arches-v4-hip-pkg-master` to simply `pkg`. Really, you can name it whatever you want.

Now go into your project's `my_project/my_project/settings.py` file and add this new line, which points to this new package, somewhere after the `APP_ROOT` line:

```
PACKAGE_DIR = os.path.join(os.path.dirname(APP_ROOT), 'pkg')
```

Note: You can actually place the package wherever you want, as long as `PACKAGE_DIR` holds the path to it. You can even leave out this setting entirely if you pass `--target path/to/package` to all of the v3 commands that are used later in this process.

Finally, load this package into your project:

```
python manage.py packages -o load_package -s pkg -db true
```

Important: We recommend using the `-db true` flag here, which will completely erase your v4 project database and create a fresh installation. If you have already added a lot of new user logins to your v4 project, these will be lost. If you have already added settings to your project like a MapBox API key, for example, follow these steps to retain them *before* running the command with `-db true`:

- In your v4 project, run `python manage.py packages -o save_system_settings`
 - Find the newly created file `my_project/my_project/system_settings/System_Settings.json` and move it into `my_project/pkg/system_settings`.
 - When you do run the load package command, say “y” to the prompt about overwriting project settings (they will be imported from this new settings file).
-

Before moving on you should be able to view your project in a browser, login with the default `admin/admin` credentials, and go to the Arches Designer to confirm that you have all six Arches-HIP Resource Models loaded. There should be no resources in your database yet.

2. Move your v3 data into the package.

Move `v3scheme-arches-<date>.xml` from *Export v3 Reference Data* into `v3data/reference_data`.

Move `v3resources-all-<date>.json` from *Export v3 Business Data* into `v3data/business_data`. This file name could be slightly different for you (or you may have multiple files) based on how you ran the v3 export.

Move `v3relations-all-<date>.csv` from *Export v3 Resource Relations* into `v3data/business_data`.

Your package should now look like this:

```
pkg/
├── v3data/
│   ├── business_data/
│   │   ├── v3resources-all-<date>.json
│   │   └── v3relations-all-<date>.csv
│   ├── graph_data/
│   ├── reference_data/
│   │   └── v3scheme-arches-<date>.xml
│   └── rm_configs.json
```


3. Convert the v3 reference data.

Run:

```
python manage.py v3 convert-v3-skos
```

New v4 reference data files will be created as shown below.

```
pkg/
├── reference_data/
│   ├── collections/
│   │   └── collections.xml
│   ├── concepts/
│   │   └── thesaurus.xml
│   └── v3topconcept_lookup.json # already existed
```

You can also add the `-i/--import` flag to automatically load the reference data into your database.

4. Convert the v3 JSON/JSONL business data.

Now you are ready to convert and import your v3 data:

```
python manage.py v3 write-v4-json
```

This command will create new v4 resource JSON/JSONL files in `pkg/business_data`, one per Resource Model. You'll be provided with easy copy/paste commands to load the files if you want, or you can add `-i/--import` to the command to load the resources immediately.

To help you debug any errors you encounter, and generally give you more control over this command, we've provided a number of optional arguments.

-i, --import	Directly imports the resources after the v4 JSON/JSONL file is created.
-m, --resource-models	List the names of resource models to process, by default all are used.
-n, --number	Limits the number of resources to load: <code>-n 10</code> will only load the first 10 resources of each resource model.
--exclude	List of resource ids (uuids) to exclude from the write process.
--only	Specify one or more resource ids to process. All other resources will be ignored.
--skipfilecheck	Skip the check for uploaded image files that are referenced in v3 business data. Only applicable if you are converting resources with images attached to them.
--verbose	Enables verbose printing during the process. Generally not recommended, it's <i>very</i> verbose.

To give a couple examples:

```
python manage.py v3 write-v4-json -m "Activity" -n 100 -i --exclude 08b68d46-c202-458a-
↪bf11-bc7a1dd5b2ef
```

will only write the first 100 “Activity” resources to v4 JSON (even if there are more Resource Models in your package), excluding a single resource whose id is `08b68d46-c202-458a-bf11-bc7a1dd5b2ef`, and will then immediately import these resources into your database.

```
python manage.py v3 write-v4-json --only 08b68d46-c202-458a-bf11-bc7a1dd5b2ef 53348d46-
↪1202-458a-bcab-fe6c7a2223cc
```

will only write the two resources matching the provided uuids.

Tip: During this process, it may be useful to use:

```
python manage.py resources -o remove_resources
```

to erase all existing resources in your database and start from scratch.

5. Convert the v3 resource relations.

Once you have all of your resources loaded in your database, you can import the resource relations from v3. Use:

```
python manage.py v3 write-v4-relations
```

to write the file, and add `-i/--import` to directly import them. You will likely get errors if you try to import resource relations but have not loaded all of your business data.

6. Load the entire package.

Though you may have been loading the individual pieces of the package along the way, the final step should be a full reload of the package.

```
python manage.py packages -o load_package -s "/full/path/to/my_project/pkg"
```

You can now treat this package just as you would any other v4 package, by adding custom functions, map layers, etc. You can also safely remove the `v3data` directory if you wish, as those files will no longer be used (generally it is good to retain that sort of data somewhere though).

App With Custom Graphs

If you have a v3 deployment with custom resource graphs, you'll need to use the following workflow. Be aware, you'll need to remake your custom resource graphs in v4 (as "Resource Models"). This is listed as Step 6 below.

Custom App Workflow

Experienced developers should be able to use some of these steps individually to accomplish discrete tasks, but we generally recommend following this workflow as a whole.

Note: All of the commands below must be run from within your v4 project.

1. Create a new package

```
python manage.py packages -o create_package -d pkg
```

The result should be a new package within your project named pkg:

```
my_project/
├─ manage.py
├─ my_project/
└─ pkg/
```

Now go into your project's `my_project/my_project/settings.py` file and add this new line somewhere after the `APP_ROOT` line:

```
PACKAGE_DIR = os.path.join(os.path.dirname(APP_ROOT), 'pkg')
```

Note: You can actually name your new package whatever you want, and place it wherever you want, as long as `PACKAGE_DIR` holds the path to it. You can even omit `PACKAGE_DIR` entirely if you pass `--target path/to/package` to all of the v3 commands below.

Finally, load this package into your project:

```
python manage.py packages -o load_package -s pkg -db true
```

Important: We recommend using the `-db true` flag here, which will completely erase your v4 project database and create a fresh installation. If you have already added a lot of new user logins to your v4 project, these will be lost. If you have already added settings to your project like a MapBox API key, for example, follow these steps to retain them *before* running the command with `-db true`:

- In your v4 project, run `python manage.py packages -o save_system_settings`
- Find the newly created file `my_project/my_project/system_settings/System_Settings.json` and move it into `my_project/pkg/system_settings`.
- When you do run the load package command, say “y” to the prompt about overwriting project settings (they will be imported from this new settings file).

Before moving on you should be able to view your project in a browser and login with the default `admin/admin` credentials.

2. Prepare your package.

```
python manage.py v3 start-migration
```

This will create some new directories and content in your package:

```
pkg/
├─ reference_data/
│   └─ v3topconcept_lookup.json
└─ v3data/
```

(continues on next page)

(continued from previous page)

```
└─ business_data/  
└─ graph_data/  
└─ reference_data/
```

3. Move your exported v3 data into the package.

Move `v3resources-all-<date>.json` from *Export v3 Business Data* into `v3data/business_data`. This file name could be slightly different for you (or you may have multiple files) based on how you ran the v3 export.

Move `v3relations-all-<date>.csv` from *Export v3 Resource Relations* into `v3data/business_data`.

Move `v3scheme-arches-<date>.xml` from *Export v3 Reference Data* into `v3data/reference_data`.

4. Move the v3 resource graph `_nodes.csv` files from v3 into your package.

In your Arches v3 deployment, you should be able to find these files in your original `source_data/resource_graphs` directory, whose contents should be a `_edges.csv` and `_nodes.csv` for every resource graph in your database. We only want the `_nodes.csv` files.

Move the `_nodes.csv` files into `v3data/graph_data`.

After completing steps 3 and 4, your v4 package should look like this:

```
pkg/  
└─ v3data/  
    └─ business_data/  
        └─ v3resources-all-<date>.json  
        └─ v3relations-all-<date>.csv  
    └─ graph_data/  
        └─ RESOURCE_GRAPH_NAME.Exx_nodes.csv  
        └─ etc.  
    └─ reference_data/  
        └─ v3scheme-arches-<date>.xml  
    └─ rm_configs.json
```

5. Convert the v3 reference data.

See *Arches-HIP workflow Step 3*, and return to this page when you've finished.

6. Build the v4 Resource Models.

Now that the v3 reference data has been converted and loaded, you are ready to create the v4 Resource Models. This migration process does not attempt to create them based on your old v3 graphs. There are a number of reasons for this, but most simply, v4 graphs have different constraints and support different datatypes and structures than those in v3. In other words, your v4 database will be better off with graphs that have been created natively, not translated from v3.

Generally, we would expect the v4 graphs to look like their v3 analogs, but we have built in quite a bit of wiggle room:

- The graph names can differ

- The node names can differ
- The graph structure can differ (though maintaining the same general branching structure is advisable)

However, there must still be a one-to-one relationship between v3 and v4 graphs and their nodes.


When it comes to node datatypes, the translation from v3 to v4 is pretty straight-forward.

Table 1: Datatype Translation – v3 to v4

v3 businesstable	v4 datatype
strings	string
dates	date or edtf
geometries	geojson-feature-collection
domains	concept - if single value per v3 branch
domains	concept-list - if multiple values per v3 branch were allowed

Important: When you set a v4 node to `concept` or `concept-list`, you will need to select which collection to use. This is why it's best to have migrated and loaded your RDM scheme (step 5 above) before making the Resource Models.

See also:

Refer to [Designing the Database](#) for help on this task. Within the Arches Designer itself, click  for detailed help on each page.

Once you have built all of the Resource Models, **export them into your package**. You can do this one-by-one from the Arches Designer interface, or use:

```
python manage.py packages -o export_graphs -d pkg/graphs/resource_models -g "all"
```

Warning: If you have made any Branches, using the `-g "all"` argument will export them as well, which you don't want. You'll have to remove them from `pkg/graph/resource_models` and/or move them into `pkg/graph/branches` before moving on.

By the end of this step, you should have one JSON file per Resource Model in `pkg/graphs/resource_models`.

7. Generate and populate the node lookup files.

Begin by running:

```
python manage.py v3 generate-rm-configs
```

which will create `v3data/rm_configs.json`. This file will be used to link the name of your v4 Resource Models with the names of their corresponding v3 graphs, as well as point to the files that link each node. Initially its content will look like:

```
{
  "Activity": {
    "v3_entitytypeid": "<fill out manually>",
```

(continues on next page)

(continued from previous page)

```
"v3_nodes_csv": "run 'python manage.py v3 generate-lookups",  
"v3_v4_node_lookup": "run 'python manage.py v3 generate-lookups"  
}  
}
```

where "Activity" is the name of a v4 Resource Model. As the file says, you must now fill out the `v3_entitytypeid` value for all items. Typically, this will look something like "ACTIVITY.E7"—upper-case with a CRM class appended to it.

Now, also as the file says, run:

```
python manage.py v3 generate-lookups
```

and you'll see the rest of the values get filled out.

There will now be more CSV files in the `v3data/graph_data` directory. There is one per v3 graph, and they are used to match the names of v3 node names (column one), with v4 node names (column two). All of the v3 nodes will be listed for you, but **you have to fill out the v4 node names manually**, using your new Resource Models for reference. A portion of a filled out file could look like:

Table 2: ACTIVITY.E7_v4_lookup.csv

v3_node	v4_node
ACTIVITY_TYPE.E55	Activity Type
ADDRESS_TYPE.E55	Address Type
etc...	etc...

Finally, you can use:

```
python manage.py v3 test-lookups
```

to check your work. Once this test passes, you can move on.

8. Convert the v3 JSON/JSONL business data

See *Arches-HIP workflow Step 4*. (You can continue using that workflow until you are finished with the migration.)

9. Write the v4 resource relations file.

See *Arches-HIP workflow Step 5*. (You can continue using that workflow until you are finished with the migration.)

10. Load the entire package (optional)

See *Arches-HIP workflow Step 6*.

1.1.4 Production Deployment of Arches on Networks

This section provides information on how to deploy Arches on a production server (including cloud hosted deployments), how to configure and manage the server, and how to manage the data and the application.

Deployment

Running Arches in “production”, as a tool for use by members of your organization or as an information source available to the public on the Web, has a few more requirements and considerations than running Arches privately on your own device.

This section documents how to set up Arches for more secure, more reliable, and more scalable production deployments. You may also choose to review documentation about *Installation with Docker*, because that also has a section about using Docker for production deployments.

Introduction to Production Deployment

This guide will walk you through the steps necessary to deploy Arches in a production environment. This guide assumes that you have already installed Arches and have a working Arches installation. If you have not yet installed Arches, please see the *Installing Core Arches*. We recommend review of [Django’s recommended checklist](#) for production deployments in order to better understand how to deploy your Arches instances in production environments.

Set DEBUG = False

Most importantly, you should never run Arches in production with `DEBUG = True`. Open your `settings.py` file (or `settings_local.py`) and set `DEBUG = False` (just add that line if necessary).

Turning off the Django debug mode will:

1. Suppress the verbose Django error messages in favor of a standard 404 or 500 error page.

You will now find Django error messages printed in your `arches.log` file.

Important: Make sure you have `500.htm` and `404.htm` files in your project’s templates directory!

2. Cause Django to stop serving static files.

You must set up a real webserver, like Apache or Nginx, to serve your app. See [Serving Arches with Apache](#).

Add Allowed Hosts and CSRF Trusted Origins to Settings

ALLOWED_HOSTS acts as a critical safeguard against HTTP Host header attacks, ensuring that your Arches application only responds to valid hostnames. On the other hand, CSRF_TRUSTED_ORIGINS is instrumental in fortifying your application against Cross-Site Request Forgery (CSRF) attacks by specifying trusted origins for the submission of forms. Both of these settings are required for Arches to work properly in production. These settings are described in more detail in the [Django documentation](#).

1. *Allowed Hosts*: In `settings.py` (sometimes set via `settings_local.py`) you will need to add multiple items to the list of `ALLOWED_HOSTS`. Consider the following example:

```
ALLOWED_HOSTS = ["my-arches-site.org", "localhost", "127.0.0.1",]
```

In that example, “my-arches-site.org” is the public domain name. But the items “localhost”, “127.0.0.1” are all local network locations where Arches is deployed. You may need all of these for Arches to work properly.

2. *CSRF Trusted Origins*: Django 4.0, a dependency of Arches 7.5 introduced a new setting for security purposes. In the `settings.py` (sometimes set via `settings_local.py`) you will need to add multiple items to the list of `CSRF_TRUSTED_ORIGINS`. If you don’t include this, users will encounter CSRF error (403) then they attempt to login. See the [Django documentation](#) for details. Note the following items (with the `https://` prefix):

```
CSRF_TRUSTED_ORIGINS = ["https://my-arches-site.org", "https://www.my-arches-site.org",]
```

Build Production Frontend Assets

In deploying Arches in production, have a choice in how you bundle frontend assets (CSS, Javascript, etc).

You can use `yarn build_development` followed by `manage.py collectstatic` to provide unminified frontend bundles. These will be larger files, so there will be a hit with respect to network performance.

Alternatively, you can build production assets for the frontend, which will be minified and therefore faster for clients to download. To make production frontend assets, use the `manage.py build_production` management command (this combines both `yarn build_production` and `manage.py collectstatic`). Please note however, you will need at least 8GB of RAM for the production frontend asset build itself (and much more if you’re also running the database and backend Arches server on the same host), and you will need lots of time. Depending on your system specifics, this can take multiple hours to complete.

Serving Arches with Apache or Nginx

Further Reference

- [Django Documentation](#)

During development, it’s easiest to use the Django webserver to view your Arches installation. However, once you are ready to put the project into production, you’ll have to use a more efficient, robust, and secure webserver like Apache or Nginx.

Use of Apache or Nginx involves many considerations in common, including set-up of SSL certificates for HTTPS, set-up and permissions of static assets, and running the Arches Django application with a WSGI server. The following guide first two sections describes how to use Apache. The next section focuses on using Nginx:

- [Configure Apache](#)
- [Prepare the Arches Project for Apache](#)

- *Configure Nginx*

Configure Apache

The following instructions work for Ubuntu 16.04 - 20.04; minor changes may be necessary for a different OS. This is a very basic Apache configuration, and more fine tuning will benefit your production installation.

1. Install Apache.

```
$ sudo apt-get install apache2
```

2. Install mod_wsgi

There are two ways to install mod_wsgi. Both of them require you to start by installing the Apache and Python development headers.

```
$ sudo apt install apache2-dev python3-dev
```

Note: You may need to install the Python dev package specific to your Python version, e.g. python3.8-dev.

Now follow one of the following two options:

Install mod_wsgi directly into your Python virtual environment

```
$ source /home/ubuntu/Projects/ENV/bin/activate
(ENV)$ pip install mod_wsgi
```

Generate the link path to mod_wsgi.

```
(ENV)$ mod_wsgi-express module-config
```

This command will produce two lines that look like

```
LoadModule wsgi_module "<your venv path>/lib/python3.8/site-packages/mod_
wsgi/server/mod_wsgi-py37.cpython-37m-x86_64-linux-gnu.so"
WSGIProxyHome "<your venv path>"
```

Copy these two lines, you will use them in step 3.

Install mod_wsgi system-wide

Alternatively, you can use apt to install at the system level:

```
$ sudo apt install libapache2-mod-wsgi-py3
```

Note that the version of Python 3 installed at the system-level may need to match the version used to create the virtual environment pointed to in the config. For example, if libapache2-mod-wsgi-py3 is compiled against Python 3.8, use Python 3.8 for your virtual environment. Installing mod-wsgi this way means you will not need to load it as a module in the Apache .conf file.

3. Create a new Apache .conf file

Here is a basic Apache configuration for Arches. If using a domain like heritage-inventory.org, name this file heritage-inventory.org.conf, otherwise, use something simple like arches-default.conf.

The paths below are based on an example project in /home/ubuntu/Projects/my_project.

```
sudo nano /etc/apache2/sites-available/arches-default.conf
```

Complete new file contents:

```
# If you have mod_wsgi installed in your python virtual environment, paste the text
↳ generated
# by 'mod_wsgi-express module-config' here, *before* the VirtualHost is defined.
LoadModule wsgi_module "/home/ubuntu/Projects/ENV/lib/python3.8/site-packages/mod_
↳ wsgi/server/mod_wsgi-py37.cpython-37m-x86_64-linux-gnu.so"
WSGIPythonHome "/home/ubuntu/Projects/ENV"

<VirtualHost *:80>

    WSGIApplicationGroup %{GLOBAL}
    WSGIDaemonProcess arches python-path=/home/ubuntu/Projects/my_project
    WSGIScriptAlias / /home/ubuntu/Projects/my_project/my_project/wsgi.py process-
↳ group=arches

    # May be necessary to support integration with possible 3rd party mobile apps
    WSGIPassAuthorization on

    ## Uncomment the ServerName directive and fill it with your domain
    ## or subdomain if/when you have your DNS records configured.
    # ServerName heritage-inventory.org

    <Directory /home/ubuntu/Projects/my_project/>
        Require all granted
    </Directory>

    # This section tells Apache where to find static files. This example uses
    # STATIC_URL = '/media/' and STATIC_ROOT = os.path.join(APP_ROOT, 'static')
    # NOTE: omit this section if you are using S3 to serve static files.
    Alias /media/ /home/ubuntu/Projects/my_project/my_project/static/
    <Directory /home/ubuntu/Projects/my_project/my_project/static/>
        Require all granted
    </Directory>

    # This section tells Apache where to find uploaded files. This example uses
    # MEDIA_URL = '/files/' and MEDIA_ROOT = os.path.join(APP_ROOT)
    # NOTE: omit this section if you are using S3 for uploaded media
    Alias /files/uploadedfiles/ /home/ubuntu/Projects/my_project/my_project/
↳ uploadedfiles/
    <Directory /home/ubuntu/Projects/my_project/my_project/uploadedfiles/>
        Require all granted
    </Directory>

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
```

(continues on next page)

(continued from previous page)

```
#LogLevel info ssl:warn
# Recommend changing these file names if you have multiple arches
# installations on the same server.
ErrorLog /var/log/apache2/error-arches.log
CustomLog /var/log/apache2/access-arches.log combined

</VirtualHost>
```

4. Disable the default Apache conf, and enable the new one.

```
$ sudo a2disssite 000-default
$ sudo a2ensite arches-default
$ sudo service apache2 reload
```

Replace `arches-default` with the name of your new `.conf` file if needed.

At this point, you can try accessing your Arches installation in a browser, but you're likely to get some kind of file permissions error. Continue to the next section.

Important: With Apache serving Arches, any changes to a `.py` file (like `settings.py`) will not be reflected until you reload Apache.

Prepare the Arches Project for Apache

1. Set all file and directory permissions.

Apache runs as the user `www-data`, and this user must have write access to some portions of your Arches project.

Note: On CentOS, Apache runs as `httpd`, so substitute that for `www-data` herein.

The `arches.log` file...

```
$ sudo chmod 664 /home/ubuntu/Projects/my_project/my_project/arches.log
$ sudo chgrp www-data /home/ubuntu/Projects/my_project/my_project/arches.log
```

The `uploadedfiles` directory...

```
$ sudo chmod 775 /home/ubuntu/Projects/my_project/my_project/uploadedfiles
$ sudo chgrp www-data /home/ubuntu/Projects/my_project/my_project/
↪uploadedfiles
```

Or, if either `arches.log` or `uploadedfiles` doesn't yet exist, you can just allow `www-data` to create them at a later point by giving write access to your project directory.

```
$ sudo chmod 775 /home/ubuntu/Projects/my_project/my_project
$ sudo chgrp www-data /home/ubuntu/Projects/my_project/my_project
```

You should now be able to access your Arches installation in a browser, but there is one more important step.

2. Run `collectstatic`.

This Django command places *all* of the static files (CSS, JavaScript, etc.) used in Arches into a single location that a webserver can find. By default, they are placed in `my_project/my_project/static`, based on `STATIC_ROOT`.

Note: You can change `STATIC_ROOT` all you want, but be sure to update the Alias and Directory info in the Apache conf accordingly.

```
(ENV)$ python manage.py collectstatic
```

The first time this runs it will take a little while (~20k files), and may show errors/warnings that you can safely ignore.

Finally, make sure Apache has write access to this static directory because *django-compressor* needs to update the *CACHE* contents inside it:

```
$ sudo chmod 775 /home/ubuntu/Projects/my_project/my_project/static
$ sudo chgrp www-data /home/ubuntu/Projects/my_project/my_project/static
```

Important: from now on, any time you change a CSS, JavaScript, or other static asset you must rerun this command.

You should now be able to view your Arches installation in a browser without any issues.

Configure Nginx

Many Django applications use the open source Nginx application as a proxy server. If you want to use nginx + uWSGI instead of Apache + mod_wsgi, you should start with [this tutorial](#). You can also use Nginx with Gunicorn (an increasingly popular way to securely run a Django application). To use Nginx and Gunicorn, please start with [this tutorial](#).

If you're using Gunicorn, don't forget to first install it into the Python virtual environment you are using for Arches:

```
$ # install gunicorn into your Arches virtual environment
$ pip install gunicorn
```

As is the case with Apache, Nginx will need appropriate permissions to serve static files. Every time you run *collectstatic*, you may change the file permissions, and you may need to rerun the following:

```
$ sudo chmod 755 /home/ubuntu/Projects/my_project/my_project/static
$ sudo chgrp nginx /home/ubuntu/Projects/my_project/my_project/static
```

It's sometimes useful to have an example configuration to help get you started. This Nginx configuration can be used as a guide.

Note: The configuration provided below asks Nginx to compress text files (css, javascript, etc). This may help to noticeably improve performance for the Arches user interface.

```
server_names_hash_bucket_size 64;
proxy_headers_hash_bucket_size 512;
server_names_hash_max_size 512;
large_client_header_buffers 8 64k;
```

(continues on next page)

(continued from previous page)

```

proxy_read_timeout 3600;
proxy_connect_timeout 3600;

# Connect to the Arches Django app running with Gunicorn.
upstream django {
    server localhost:8000;
}

# The not encrypted plain HTTP config
server {
    listen 80;
    charset utf-8;
    server_name my-arches-project.org www.my-arches-project.org;

    location ^~ /.well-known/acme-challenge/ {
        default_type "text/plain";
        autoindex on;
        allow all;
        root /var/www/certbot/$host;
    }

    access_log /logs/nginx/access.log;
    error_log /logs/nginx/error.log;
    proxy_read_timeout 3600;

    proxy_set_header    X-Forwarded-Protocol  $scheme;
    gzip on;
    gzip_disable "msie6";
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_buffers 16 8k;
    gzip_http_version 1.1;
    gzip_types text/plain text/css application/json application/ld+json
application/geo+json text/xml application/xml application/xml+rss
text/javascript application/javascript text/html;

    # Redirect to use HTTPS
    location / {
        return 301 https://$host$request_uri;
    }
}

# The encrypted HTTPS config
server {
    listen 443 ssl;

    server_name my-arches-project.org www.my-arches-project.org;
    access_log /logs/nginx/ssl_access.log;
    error_log /logs/nginx/ssl_error.log;

    proxy_set_header    X-Forwarded-Protocol  $scheme;

```

(continues on next page)

(continued from previous page)

```

proxy_read_timeout 3600;

ssl_certificate /etc/your-ssl-path/fullchain.pem;
ssl_certificate_key /etc/your-ssl-path/privkey.pem;

# NOTE! These other config files are not documented here
include /etc/nginx/options-ssl-nginx.conf;
ssl_dhparam /etc/nginx/sites/ssl/ssl-dhparams.pem;
include /etc/nginx/hsts.conf;

# NOTE! Be default, NGINX only allows a 1MB file upload.
# The following config raises this to 100MB
client_max_body_size 100M;

# Ask Nginx to use gzip compression to send javascript, css, etc.
gzip on;
gzip_disable "msie6";
gzip_vary on;
gzip_proxied any;
gzip_comp_level 6;
gzip_buffers 16 8k;
gzip_http_version 1.1;
gzip_types text/plain text/css application/json application/ld+json
application/geo+json text/xml application/xml application/xml+rss
text/javascript application/javascript text/html;

location ^~ /.well-known/acme-challenge/ {
    default_type "text/plain";
    autoindex on;
    allow all;
    root /var/www/certbot/$host;
}

# For the 'alias', use the correct path to the location where Arches
# puts static files after 'collectstatic'. Like Apache (see above)
# Nginx will also need permissions to serve the static files.
location /static/ {
    autoindex on;
    allow all;
    alias /path_to_arches_static_files_after_collectstatic/;
    include /etc/nginx/mime.types;
}

location @proxy_to_django {
    proxy_pass http://django;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_redirect off;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

```

(continues on next page)

(continued from previous page)

```
    proxy_set_header X-Forwarded-Host $server_name;
}
}
```

Implementing SSL

Secure Sockets Layer (SSL) enables the server to establish an encrypted link with its clients. This is more secure than using unencrypted communication. To implement SSL you will need a digital certificate which can be signed either by you or by a certificate authority. For more information about the SSL certificates please see [this article](#) .

Implementing SSL on your server can be divided to two stages:

- *Obtaining a SSL certificate*
- *Configuring the webserver*

Obtaining a SSL certificate

SSL certificate can be signed either by you using your own private key or by a certificate authority. Each choices has its own prerequisites and consequences.

Self signed

A good guide about how to implement this using OpenSSL on Ubuntu 20.04 can be found [here](#).

Note: This option allows you to implement SSL using your server's IP address without a domain name. However, when accessing the website using any modern browser the connection will be marked as not private.



Your connection is not private

Attackers might be trying to steal your information from Your Server's IP (for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR_CERT_COMMON_NAME_INVALID



To get Chrome's highest level of security, [turn on enhanced protection](#)

Advanced

Back to safety

Signed by Let's encrypt

Let's Encrypt is a non-profit certificate authority run by Internet Security Research Group that provides X.509 certificates for Transport Layer Security encryption at no charge. You can obtain a certificate from other certificate authorities too. Please keep in mind that some authorities require a fee for their services.

Note: For this option you will need a domain name to use for your website.

Install certbot

certbot is a tool that helps you obtain a certificate from Let's encrypt. [The official installation instructions for apache running on Ubuntu 20.04 can be found here.](#)

Configuring the webserver

To use the digital certificate in serving your website you need to modify the webserver configuration. You can modify the current configuration file to add the new configuration or create a new configuration file. In this guide we will use one file.

Start by adding the domain as a variable at the top of the file as such

```
ServerName yourDomainName
```

Then modify the current configuration to redirect the requests from port 80 to port 443. You will need to add this code

```
RewriteEngine On
RewriteCond %{SERVER_PORT} !^443$
RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R=301,L]
```

You can transfer all the configuration related to arches to the new virtual host 443 and change `</path/to/your/certificate/>` to reflect the location of your certificate. Your file should look like this

```
ServerName yourDomainName
LoadModule wsgi_module "/home/ubuntu/Projects/ENV/lib/python3.8/site-packages/mod_wsgi/
↳server/mod_wsgi-py37.cpython-37m-x86_64-linux-gnu.so"
WSGIPythonHome "/home/ubuntu/Projects/ENV"
<VirtualHost *:80>
    ServerName yourDomainName
    ServerAdmin webmaster@localhost

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # This is optional, in case you want to redirect people
    # from http to https automatically.
    RewriteEngine On
    RewriteCond %{SERVER_PORT} !^443$
    RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R=301,L]

</VirtualHost>
<VirtualHost *:443>
    WSGIPassAuthorization on
    WSGIDaemonProcess arches python-path=/home/ubuntu/Projects/my_project
    WSGIScriptAlias / /home/ubuntu/Projects/my_project/my_project/wsgi.py process-
↳group=arches
    <Directory /home/ubuntu/Projects/my_project/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
```

(continues on next page)

(continued from previous page)

```
Alias /media/ /home/ubuntu/Projects/my_project/my_project/static/
<Directory /home/ubuntu/Projects/my_project/my_project/static>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

Alias /files/uploadedfiles /home/ubuntu/Projects/my_project/my_project/uploadedfiles
<Directory /home/ubuntu/Projects/my_project/my_project/files/uploadedfiles>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

ServerName yourDomainName
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

SSLEngine on
SSLCertificateFile </path/to/your/certificate/>cert.pem
SSLCertificateKeyFile </path/to/your/certificate/>privkey.pem
SSLCACertificateFile </path/to/your/certificate/>chain.pem
</VirtualHost>
```

Then you will need to enable the SSL and redirecting modules before you reload apache configuration

```
sudo a2enmod ssl
sudo a2enmod rewrite
```

Now you can reload apache to access the new configuration

```
sudo service apache2 reload
```

Setting up Supervisor for Celery

Arches uses Celery (<https://docs.celeryq.dev/en/stable/getting-started/introduction.html>), a Python framework for setting up and managing task queues. Using Celery, Arches can delegate certain tasks with long execution times to separate processes. In a production deployment, this can enable Arches to delegate big jobs to a queue so that requests to the Arches application do not lead to timeout or other errors.

This documentation discusses how to enable Celery task management using Supervisor (<http://supervisord.org/>). Essentially, Supervisor automatically monitors and controls Celery workers, checking to make sure they are operating, and restarting them if they fail.

When Do I Need Supervisor and Celery?

Arches *does not require* Supervisor and Celery to run in “production” mode (with `DEBUG=False` in `settings.py`). Arches instances managing smaller amounts of data may not need Supervisor and Celery. The deployment scenarios where you *should* consider using Supervisor and Celery include:

- Supervisor and Celery will be required if you want to enable export / bulk download of more than 2000 resource instances.
- Supervisor and Celery will be required to enable the Bulk Data Manager plugin to function. (Note: by default, Arches installs this plugin in hidden state.)

Supervisor and Celery Installation and Configuration

The following is a guide **for a linux-based** OS; be advised you can change any of the file names, destinations, or permissions to suit your needs.

1. Supervisor can be installed using in your Arches virtual environment with pip: `pip install supervisor`.
2. In the core arches repo, in `arches/install/supervisor_celery_setup` there exist example files for supervisor, celeryd, and celerybeat. We recommend copying them into the following directory structure:

```
/etc/supervisor/
|-- my_proj_name-supervisord.conf
|-- conf.d/
|   |-- my_proj_name-celeryd.conf
|   |-- my_proj_name-celerybeat.conf
```

3. In the content of the files as well as the filenames themselves, replace the values of the following placeholders:
 - `/absolute/path/to/virtualenv/` - absolute path to your python3 virtualenv
 - `[app]` - replace this with the value of `ELASTICSEARCH_PREFIX` in your project’s `settings.py` file
 - `/absolute/path/to/my_proj` - absolute path to your arches project
 - `my_proj_name` - name of your project
4. Note that you can change the value for `user` in the `-supervisord.conf` file to a designated user to run supervisor.
5. Before proceeding, you will want to make sure that whichever user you designate to run supervisor has the appropriate permissions for the following files:
 - `/var/log/supervisor/supervisord.log`
 - `/var/log/celery/worker.log`
 - `/var/log/celery/beat.log`
 - `/var/run/supervisord.pid`
6. Download and install RabbitMQ: <https://www.rabbitmq.com/download.html>
7. Once successfully installed (and verified that it has been added to your `PATH`), start running it with the command `rabbitmq-server`. For a convenient option, this can be run in a screen. Note that rabbitmq should be run prior to running supervisor. If you choose, you can use Redis as a “broker” instead of RabbitMQ (see below).
8. Run `supervisord -c /etc/supervisor/my_proj_name-supervisord.conf` to start the supervisor which will start celery workers for your tasks.
9. To check and stop your supervisor process, please review the following:

- a. To check on the status of celery (workers): `supervisorctl -c /etc/supervisor/my_project-supervisor.conf status`
- b. To restart celery workers: `supervisorctl -c /etc/supervisor/my_project-supervisor.conf restart celery`
- c. To stop celery workers: `supervisorctl -c /etc/supervisor/my_project-supervisor.conf stop celery`
- d. To shut down supervisord (and the celery processes it controls): `unlink /tmp/supervisor.sock`

Setting up Redis Instead of RabbitMQ

Redis can serve as an alternative to RabbitMQ, but it lacks official Windows support. If you are not deploying Arches on Windows, you can use Redis as follows:

1. Follow the above directions (steps 1 to 5) for setting up supervised, celery, and their configurations
2. Install Redis (see: <https://redis.io/docs/getting-started/>)
3. Install the Python interface to Redis into your Arches virtual environment with pip: `pip install redis`
4. Configure the `CELERY_BROKER_URL` (in `settings.py` or overwritten in `settings_local.py`):
`CELERY_BROKER_URL = "redis://@localhost:6379/0"`
5. Activate Redis: `redis-server`
6. Run `supervisord -c /etc/supervisor/my_proj_name-supervisord.conf` to start the supervisord and celery workers
7. Start Arches

Known Issue with Arches Celery Configurations and Celery Beat

The default configuration files in the `conf.d` directory discussed above need updating. Version 5 and later of celery has a revised order of arguments in using celery commands (see: <https://github.com/archesproject/arches/issues/9202>).

- The corrected syntax (celery >= v5x) for the command at the top of `my_proj_name-celeryd.conf` looks like: `/absolute/path/to/virtualenv/bin/celery -A my_proj_name.celery worker --loglevel=INFO`
- The corrected syntax (celery >= v5x) for the command at the top of `my_proj_name-celerybeat.conf` looks like (all in one line): `/absolute/path/to/virtualenv/bin/celery -A my_proj_name.celery beat --schedule=/tmp/celerybeat-schedule --loglevel=INFO --pidfile=/tmp/celerybeat.pid`

After fixing the command syntax, the celery worker should function. However, you may still have trouble getting celery beat to work (<https://github.com/archesproject/arches/issues/9243>). Celery beat schedules periodic tasks (much like a *crontab* in a Linux operating system) using a Python implementation. In many cases, Arches will function without (evident) problems even if celery beat does not work. However, if you have workarounds or fixes, please let us know!

Cron Rebooting Start Problem and Workaround

You may encounter a problem if you use a @reboot cron job to start up Supervisor as described in the docs. This may lead to connection errors because Celery can't reach RabbitMQ. One workaround that may help would be to wait a minute or two, and then rerun that same startup command. This will hopefully allow RabbitMQ enough time to be ready to accept connections with Supervisor and Celery.

More information

- Supervisor documentation: <http://supervisord.org/>
- Celery sample files for supervisord: <https://github.com/celery/celery/tree/master/extra/supervisord>
- Redis: <https://redis.io/>
- Redis (Python interface): <https://pypi.org/project/redis/>

Backing up the Database

<https://github.com/archesproject/arches-docs/issues/132>

Using AWS S3 or Other Cloud Storage

By the time you are in a production environment, you will have configured Arches with a web server, such as Apache or nginx. While you need a web server to serve the app itself, there are two pieces of the app that can be separated from the web server and served independently. These are the 'static' files (the css, javascript, and logos that are used throughout the app) and the 'media' files (any user uploaded files, such as images or documents).

Why Use Cloud Storage (Like S3) with Arches?

These static and media files need to be stored someplace accessible via Web (HTTP) requests made by Arches users. Many of the existing tutorials on this matter are concerned with serving both static and media files, because the more load you can take off of your web server the better. However, for the purposes of this tutorial, we are only dealing with media files. S3 (and other cloud storage services!) are especially suited to storing a large (and growing) amount of files. For instance:

- Cloud storage is cheap: As per the [S3 price chart](#), it costs just \$.03 per gb/month. So a database with 10gb of photos will have a media storage cost of \$3/month, plus a small amount per transaction (\$.004 per 10,000 GET requests, e.g.). [Google Cloud storage](#) has similar costs, as does [Azure Cloud storage](#).
- Cloud storage is scalable: You only pay for the amount of data you have stored, and you have no real limit on how much you can store. This allows for an Arches deployment on a small server, either in-house or a small cloud instance (AWS EC2, Google, DigitalOcean, etc.) to store hundreds of gigabytes of media—photos, audio, video, documents—without having to restructure to accommodate more data.

You should be able to use Cloud storage regardless of where your app is hosted, whether on an internal server, an AWS EC2 instance, a DigitalOcean droplet, etc.

Note: We provide specific guidance for integrating Arches with Amazon S3 storage because it is currently popular and familiar to many. However, we want to emphasize that you can choose among different commercial cloud storage services to use with Arches. The S3 integration steps below will give you a general picture on how to use other cloud

storage services, but you'll need to change some specifics. Please refer to [django-storages documentation](#) for additional help on integrating with different cloud storage providers.

Note: We've found that by following the steps below, deleting an Information Resource from within Arches will *not* automatically remove the file from your S3 bucket. You can manually delete files from the bucket for now, or the intrepid developer may check out the answer to [this question](#) on the Arches forum.

Warning: You may run into some version compatibility issues with Arches, Django, and [django-storages](#). If your version of Arches uses a version of Django that is <3.2, pip installing django-storages will install the latest version of django (incompatible with Arches) and cause your Arches application to break. If you run into this problem, you may need to use pip to reinstall the Arches requirements as specified in the Arches *requirements.txt* file.

Steps to Follow

To use S3, you will need an AWS account, which is just an extension of a normal Amazon account. [Here's](#) some information on how to get started.

Having worked through a number of existing tutorials (mostly [dylanbfox.blogspot.com](#), [www.caktusgroup.com](#), and [www.holovaty.com](#)), we've distilled these steps to show how you can use S3 in conjunction with your Arches app. Before beginning, you will need to have set up and logged into your AWS account.

1. Create credentials for your Arches app

These new credentials will allow your Arches app to access the S3 bucket.

1. Access the AWS Identity and Access Management (IAM) Console.
2. Create a new user (named something like "arches_media"), and download the new credentials. This will be a small .csv file that includes an Access Key ID and a Secret Key.
3. Also, go to the new user's properties, and record the User ARN.

2. Create a new bucket on S3

Next, you'll need to create a new bucket and give it the appropriate settings.

1. Create a bucket, named something like "my_app-media".
2. In the new bucket properties, under Permissions, create a new bucket policy
3. Paste the following text into your new policy, inserting your own BUCKET-NAME and the your new User ARN

```
{
  "Statement": [
    {
      "Sid": "PublicReadForGetBucketObjects",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": ["s3:GetObject"],
      "Resource": ["arn:aws:s3:::BUCKET-NAME/*"]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

    },
    {
      "Action": "s3:*",
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3:::BUCKET-NAME",
        "arn:aws:s3:::BUCKET-NAME/*"
      ],
      "Principal": {
        "AWS": [
          "USER-ARN"
        ]
      }
    }
  ]
}

```

4. Also, make sure that the CORS configuration (click “Add CORS Configuration”) looks like this

```

<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>*/</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
    <AllowedHeader>Authorization</AllowedHeader>
  </CORSRule>
</CORSConfiguration>

```

3. Update the Virtual Environment

In order to configure Arches to use your new bucket, you need to install a couple of extra Django modules in your virtual environment. These will augment Django’s flexibility in how it stores uploaded media.

Activate your virtual environment and run this command

```
(ENV) $: pip install boto3==1.26 django-storages==1.13
```

4. Update *settings.py*

Finally, you need to tell your app to use these new modules, give it the necessary credentials, and tell it where to store (and find) the uploaded media. Open the your *settings.py* file...

1. Find the line that defines the settings “INSTALLED_APPS” and add ‘storages’ to it. It should look like this

```
INSTALLED_APPS = INSTALLED_APPS + (PACKAGE_NAME, 'storages',)
```

2. Next, add the following lines, replacing the AWS settings values with information from earlier steps (remember the *credentials.csv* file you downloaded?)

```

STORAGES = {
    "default": {
        "BACKEND": "storages.backends.s3boto3.S3Boto3Storage",
    },

```

(continues on next page)

(continued from previous page)

```
"staticfiles": {
    "BACKEND": "django.contrib.staticfiles.storage.
StaticFilesStorage",
},
}
AWS_STORAGE_BUCKET_NAME = 'aws_bucket_name'
AWS_ACCESS_KEY_ID = 'aws_access_key_id'
AWS_SECRET_ACCESS_KEY = 'aws_secret_access_key'
S3_URL = 'http://%s.s3.amazonaws.com/' % AWS_STORAGE_BUCKET_NAME
MEDIA_URL = S3_URL
```

3. Restart your web server.

You should be good to go! To test, create a new Information Resource in your installation and upload a file. Now go back to check out your S3 bucket through the AWS console. Your file should show up in a new folder called files within the bucket. If you are encountering issues, be sure to let us know on the [forum](<https://groups.google.com/forum/#!forum/archesproject>).

Migrating a Local App to AWS EC2

If you've been doing your Arches development work locally you will eventually need to transfer your app to a remote server of some kind in order for it to be served through the Internet. This can be done in many different ways, and in this section we'll give an introductory explanation of how to use Amazon Web Services (AWS) to deploy Arches.

Overview

AWS includes dizzying array of systems and services. AWS names different computing services using an alphabet soup of (initially) cryptic acronyms. Acronyms mentioned in this documentation include:

AWS:

Amazon Web Services

ALB:

Application Load Balancer (ALB) provides (in this context) a means to manage requests from the outside Internet before they get directed to your EC2 instance running Arches.

EC2:

Amazon Elastic Compute Cloud (EC2) provides virtual servers with different processing speeds, memory, hard drives, and operating systems. You can install your own software (such as Arches) on EC2 instances.

RDS:

Amazon Relational Database Service (RDS) provides Amazon managed database servers (including Postgres servers) that you can use instead of manually installing and managing a database server on an EC2 instance.

S3:

Amazon Simple Storage Service (S3) provides a scalable file storage and hosting infrastructure.

IAM:

Amazon Identity and Access Management (IAM) service sets security and permissions roles and policies for different users, different applications and services, and different computing instances.

While it can be very intimidating to get started, Amazon Web Services are so widely used that you can easily find some excellent guidance and help.

This guidance is primarily intended to provide a basic introduction for simple AWS deployment architectures. Some organizations use AWS for relatively small-scale and simple projects, and others use AWS to run large-scale and very complicated systems. An AWS deployment architecture may vary widely according to different security requirements, scales, backup strategies, maintainability needs, and uptime and performance needs.

To help get you started, this documentation focuses on simple initial AWS deployments. An organization should consult with AWS experts in cases where there are significant security, scale, reliability, or performance requirements. Please look elsewhere for guidance on server administration and maintenance.

Example Deployment Architectures

As noted above, you should carefully align your deployment architecture according to your specific requirements, budget, and proficiency with AWS services. This introduction illustrates just two of a wide variety of architecture options:

1. A Single Node Deployment (one EC2 instance)

The most simple AWS deployment architecture essentially mimics deployment of Arches that runs as a localhost on your own machine. In this architecture, the Arches application runs on a single EC2 instance *along with* the dependency Postgres database server and dependency Elasticsearch server. As described in the diagram below, the only other AWS service used outside of this one EC2 instance is S3 (to configure, see: [Using AWS S3 or Other Cloud Storage](#)), for storage for user uploaded files.

2. A Multiple Node Deployment (two EC2 instances and RDS)

If you require greater performance, you can consider an architecture that uses multiple EC2 instances together with other AWS services, especially the RDS service. For example, you can deploy the Elasticsearch server (an Arches dependency) on a separate EC2 instance. This avoids a scenario where the Arches core application and Elasticsearch compete for the same computing resources. Similarly, you can use the RDS service to provision a Postgres database for Arches, which again more widely distributes computation across a broader infrastructure. Using multiple EC2 instances together with the RDS service may be somewhat more expensive and may involve a bit more configuration and deployment effort, but this architecture will likely have scale and performance advantages. As described in the diagram below, (core) Arches and Elasticsearch each have their own EC2 instances, RDS provisions the Postgres database to Arches, and S3 provides storage for user uploaded files (to configure, see: [Using AWS S3 or Other Cloud Storage](#)).

AWS Security and Permissions Management

AWS provides extremely powerful and sophisticated tools to manage permissions and security. AWS emphasizes the management of “roles” and “policies” for security. You typically use the IAM service to set roles and policies that grant specific permissions to individuals or services. A good security practice is to follow “the principle of least privilege”. This principle ensures that entities only have the bare minimum permissions necessary to perform their tasks.

If you are managing sensitive information in Arches (or any other system) you should gain proficiency with AWS security good practices and a good understanding of network architecture. For example, if you deploy Arches using an EC2 instance on a public subnet, SSH access will be more convenient, but it will be less secure than putting the Arches EC2 instance in private subnet. The best choice of security practices and network protections will vary depending on the sensitivity of the information you manage and your operational / administrative needs.

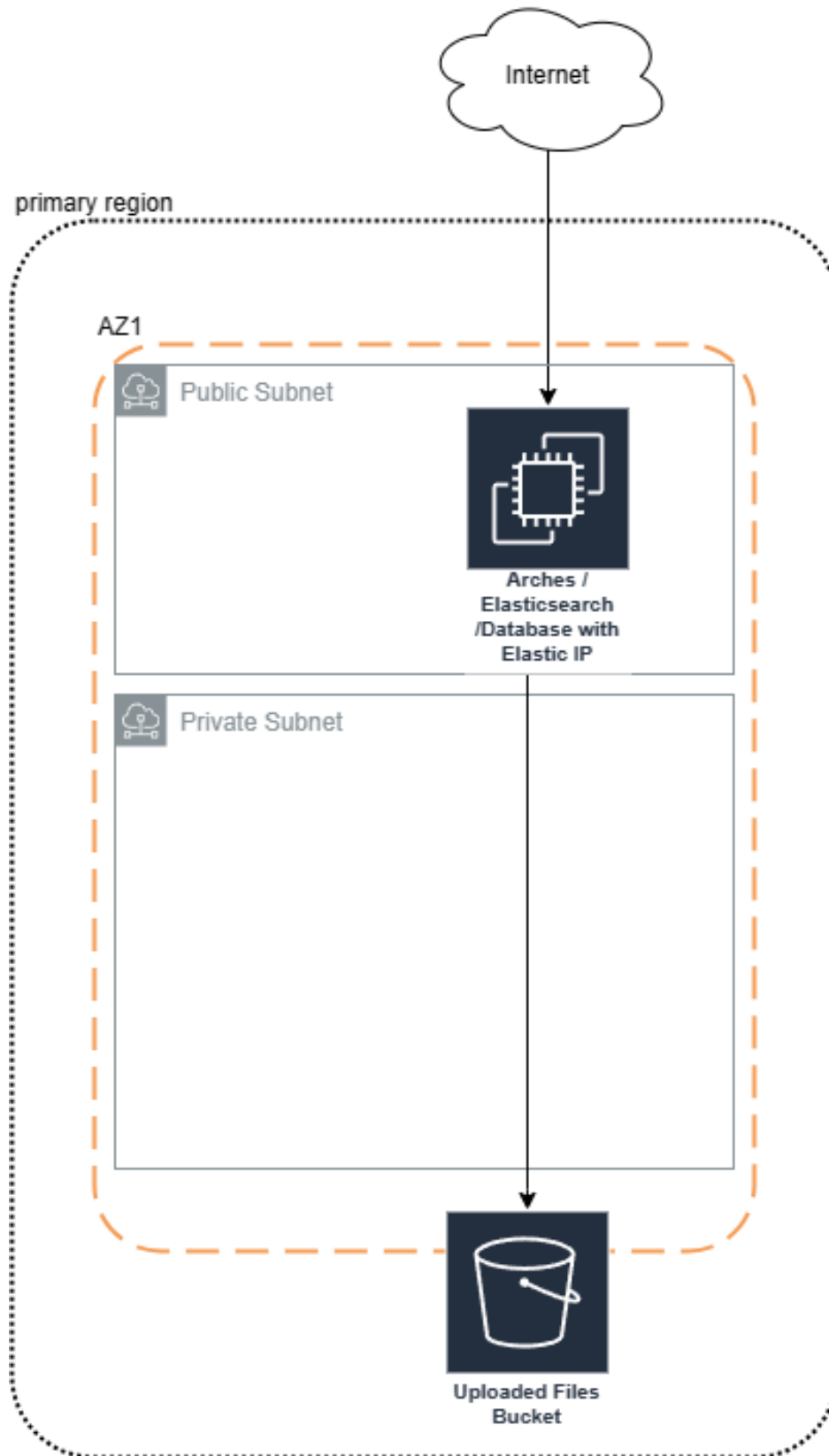


Fig. 44: Arches deployed on a single EC2 instance.

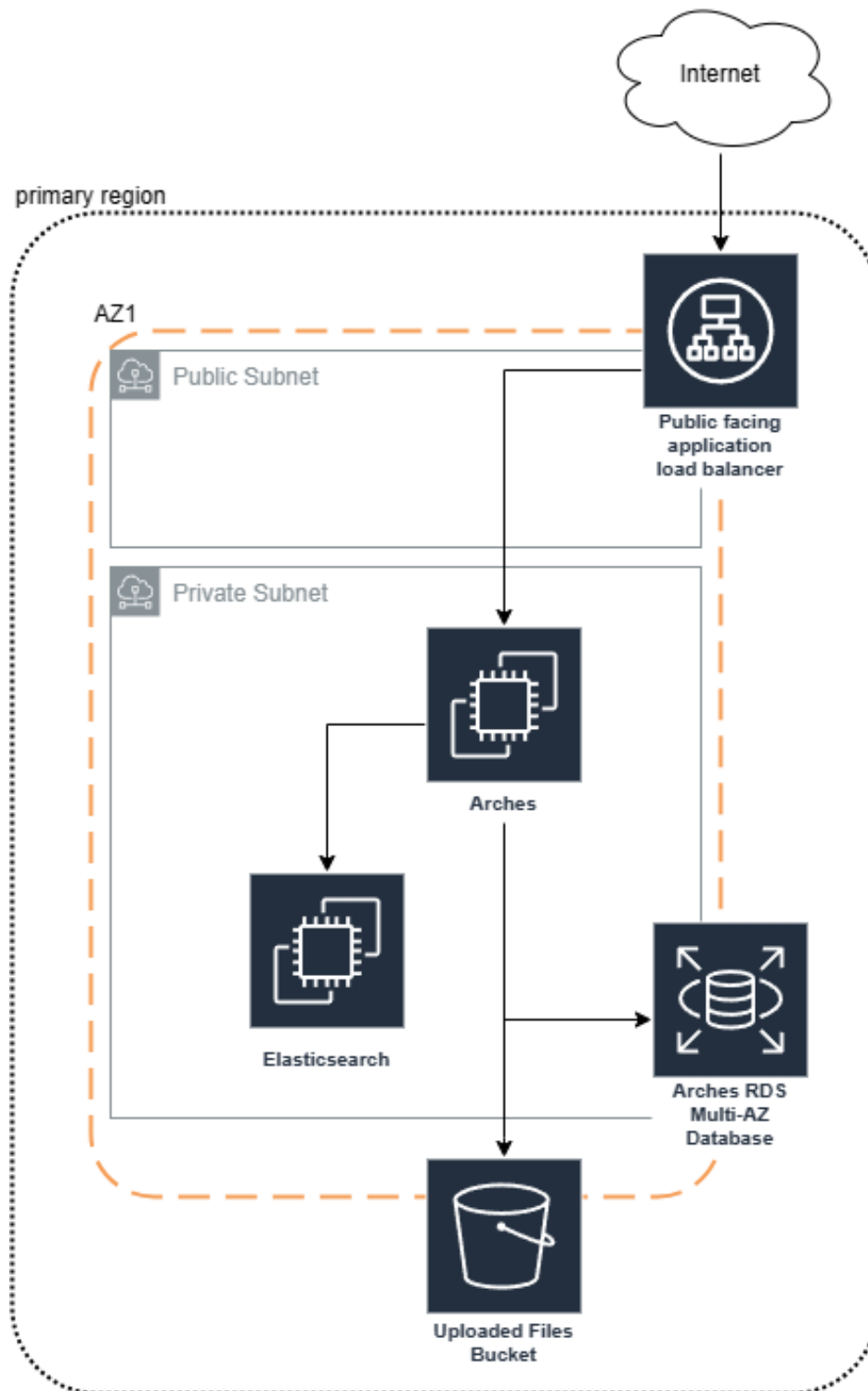


Fig. 45: Arches deployed on multiple EC2 instances together with RDS.

Moving a Localhost Arches Project to EC2

Now that we've introduced different considerations for deploying Arches on AWS, we can move into more specifics about how to move a locally hosted Arches instance to a "Single Node Deployment" (see the architecture described above).

Warning: Some content below may be outdated. Both Arches and AWS are evolving systems. If you notice sections that need updating please alert us by submitting a ticket (<https://github.com/archesproject/arches-docs/issues>)!

Prerequisites

A few components must be in place before you are ready to complete these steps.

1. You will need an [AWS account](#), which is just an extension of a normal Amazon account. In the very beginning, do not worry about pricing; if you are new to AWS, everything listed below will fall in the "free tier" for one year.
2. You'll need an SSH client in order to access your remote server's command console. For Windows, we recommend [PuTTY](#) as an easy to use, light-weight SSH client. While downloading PuTTY, also be sure to get its companion utility, PuTTYgen (from the same webpage).
3. You'll need an FTP client in order to transfer files (your Arches app customizations) to your server. We recommend [FileZilla](#).

Once you have an AWS account set up, and PuTTY/PuTTYgen and FileZilla installed on your local computer, you are ready to begin.

Note: Experience with command line tools, especially those that involve the management of security (encryption) certificates (such as `ssh` and `scp`) is typically necessary to deploy and manage Arches on remote cloud computing services.

Create an EC2 Instance

From your AWS account console, navigate to the EC2 section. You should get to a screen that looks something like this:

Click on "Launch Instance"

You now have the opportunity to customize your instance before you launch it, and you should see seven steps listed across the top of the page. For our purposes, we only need to worry about a few of them:

- In Step 1, choose "Ubuntu Server 22.04 LTS" as your operating system
- In Step 2, choose an instance type
- In Step 3, tag your server with a name (this is helpful, though not necessary)
- **In Step 4, you'll need to:**
 - Select "Create a new security group"
 - Name it "arches-security"

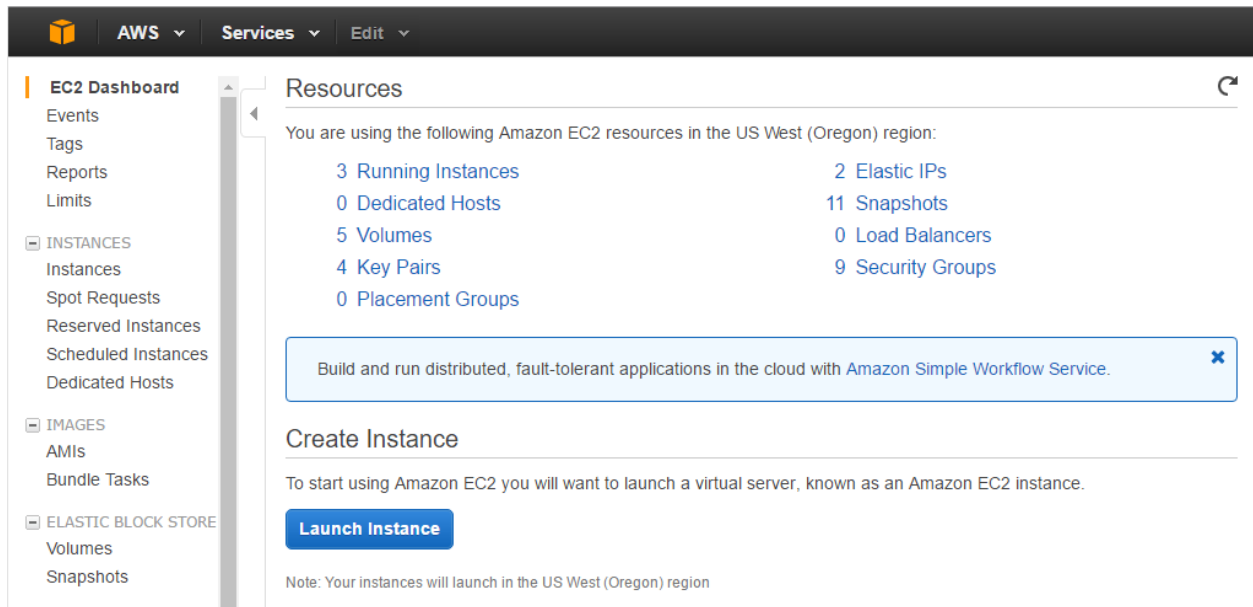


Fig. 46: A (dated) view of the EC2 dashboard (AWS dashboard interfaces frequently change)

- Modify the rules of this security group to match the following

Type	Protocol	Port Range	Source
HTTP	TCP	80	Anywhere (0.0.0.0/0)
HTTPS	TCP	443	Anywhere (0.0.0.0/0)
Custom TCP Rule	TCP	8000	Anywhere (0.0.0.0/0)
SSH	TCP	22	My IP

- In Step 5, click Launch

When you launch the instance, you will be asked to create a new key pair. This is very important. Name it something like “arches-keypair”, and download it to an easy-to-access location on your computer. You will use this later to give the SSH and FTP clients access to your server. **Do not misplace this file.**

Once you have launched the instance, click “View Instances” to see your running (and stopped) EC2 instances. The initialization process takes a few moments, so we can leave AWS alone for now and head to the next step.

NOTE Your Security Group is the firewall for your server. Each rule describes a specific type of access to the server, through a specific port, from a specific IP address. *Never* allow access through port 22 to any IP but your own. If you need to access your server from a new location (library, university) you’ll need to update the SSH security rule with your new IP address.

Convert your AWS .pem Key Pair to a .ppk Key Pair

PuTTY uses key files in a different format than AWS distributes by default, so you'll have to make a quick conversion:

1. Open PuTTYgen
2. Click Load
3. Find the .pem file that you downloaded when launching your instance (you may have to switch to "All Files (.)")
4. Once loaded, click Save
5. Ignore the prompt for a passphrase, and save it with the same name as your original .pem file, now with the .ppk extension.

Connect to your EC2 Instance with PuTTY

Now go back to AWS, and look at the status of your server instance. By now, it probably says "2/2 checks passed" in the Status Checks column, and you should have an address (xx.xx.xx.xx) listed in the Public IP column. It's ready!

1. Open PuTTY, and enter your server's Public IP into the Host Name bar. Make sure Port = 22, and the Connection Type is SSH (remember the security rules we were working with?).
2. To make PuTTY aware of your key file, expand the SSH section in the left pane, and click on Auth. Enter your .ppk file as the "Private key file for authentication".
3. Once you have the IP Address and key file in place, click Open.
4. Click OK to trust the certificate, and login to your server as the AWS default user *ubuntu*.
5. If everything goes well you should be greeted with a screen like this:

```
Authenticating with public key "imported-openssh-key"
Welcome to Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-48-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Fri May 13 17:11:11 UTC 2016

System load:  0.0                       Processes:    111
Usage of /:   70.1% of 15.61GB           Users logged in:  0
Memory usage: 45%                       IP address for eth0: 172.31.10.110
Swap usage:   0%

Graph this data and manage this system at:
  https://landscape.canonical.com/

Get cloud support with Ubuntu Advantage Cloud Guest:
  http://www.ubuntu.com/business/services/cloud

7 packages can be updated.
7 updates are security updates.

Last login: Fri May 13 17:11:16 2016 from 98.143.235.121
ubuntu@ip-172-31-10-110:~$
```

Congratulations! You've successfully navigated your way into a functional AWS EC2 instance.

Install Arches Dependencies on your EC2 Instance

Now that you have a command line in front of you, the next few steps should be very familiar. Luckily, if you are coming from Windows, you'll find that installing dependencies on Ubuntu is much, much easier. Do all of the following from within the `/home/ubuntu` directory (which shows as `~` in the command prompt).

1. Download the install script for dependencies (this links to the v7.5.0 dependencies install, update the link for your specific version)

```
$ wget https://raw.githubusercontent.com/archesproject/arches/stable/7.5.0/
↪arches/install/ubuntu_setup.sh
```

2. Run it (this may take a few minutes)

```
$source ./ubuntu_setup.sh
```

Install Arches and on your EC2 Instance

There's no hard and fast rule about where in the filesystem you should install Arches on an EC2 instance. A typical deployment scenario would be to install Arches on an Ubuntu EC2 instance. For that kind of instance, Amazon will provide SSH credentials to log in as the `ubuntu` user (with super user privileges). That means when you login via SSH to your Arches EC2 instance, you should find yourself in the `/home/ubuntu` directory.

For sake of simplicity and consistency, we'll assume you will be installing Arches within the `/home/ubuntu` directory. However, you may choose an alternate location like a sub-folder of the `/opt` directory. The `/opt` directory may be more convenient if you want to make Arches easier to manage by multiple people with different user accounts.

Once you have the dependencies installed, see [Installing Core Arches](#), you can copy your Arches project from your local machine to the desired location on your Arches EC2 instance. You can use Filezilla to do that, or use the command-line utility `scp`.

Connect to your EC2 Instance with Filezilla

To transfer files from your local environment to your EC2 instance, you'll need to use an FTP client. In this case we'll use FileZilla.

First, we'll need to set up the authentication system to be aware of our AWS key file.

1. Open FileZilla
2. Go to Edit > Settings > SFTP and click Add key file...
3. Navigate to your `.ppk` file, and open it. You'll now see you file listed.
4. Click OK to close the Settings

Next, you can use the "Quickconnect" bar:

- Host = your server's Public IP
- Username = ubuntu
- Password = <leave blank> (that's what the `.ppk` file is for)
- Port = 22

Once connected, you'll see your server's file system on the right side, and your local file system on the left. Find your local `"my_hip_app"` directory, and copy the entire directory to `/home/ubuntu/Projects/`. This example directory

structure is consistent with related documentation explaining how to set up Apache or Nginx for use with Arches, see: [Serving Arches with Apache or Nginx](#)

Next, use this command to remove the elasticsearch installation from your new app on the server (because ElasticSearch should be *installed* on on your EC2 instance).

```
(ENV)$ sudo rm -r my_hip_app/my_hip_app/elasticsearch
```

Now you can run these final commands from the `/home/ubuntu/Projects/my_hip_app` directory to complete your app's installation on the server:

1. Install ElasticSearch

```
(ENV)$ python manage.py packages -o setup_elasticsearch
```

2. Run ElasticSearch

```
(ENV)$ my_hip_app/elasticsearch/elasticsearch-8.5.3/bin/elasticsearch -d
```

3. Install the app

```
(ENV)$ python manage.py packages -o install
```

4. Run the devserver

```
(ENV)$ python manage.py runserver 0:8000
```

Note: In this case, explicitly setting the host:port with `0:8000` ensures that the server is visible to us when we try to view it remotely.

You should now be able to open any web browser and view your app by visiting your IP address like so: <http://xx.xx.xx.xx:8000>. Now that you have transferred your app to a remote server, its time to use a real production-capable webserver like Apache to serve it (that's how we can get rid of the `:8000` at the end of the url). If you can't see Arches, check AWS networking permissions to make sure port `8000` is accessible. But once you've verified Arches is working, *DO NOT* leave port `8000` open. Leaving it open will be a security risk.

Another way to check would be SSH onto your Arches EC2 instance and use curl to see if Arches is responding.

```
curl http://localhost:8000
```

If the above command gives you raw HTML, then Arches is functioning and responding to requests to port 8000.

Keep in mind that you may need to have different values in your `settings.py` file once you have transferred it to a new operating system (GDAL_PATH, for example). To handle this, create and use a different `settings_local.py` file on each installation.

More Advanced Configurations

As noted above AWS security management can be complex. It is best to consult with experts in AWS to get advice about your specific deployment scenario. Generally speaking, when implementing a “Multiple Node Deployment” (see above) architecture, you should set up a unique (and clearly named) security group for each EC2 instance and the RDS instance involved in your deployment. You can then set the minimum required “inbound” rules that allow members of each of these security groups to connect as needed. For example, an EC2 instance running ElasticSearch would have its own security group. That ElasticSearch security group would have an inbound rule that allows connections from the Arches EC2 instance security group at the desired port (the default port for client, like Arches, API calls connecting with ElasticSearch is 9200).

Some additional (advanced) considerations include:

1. *RDS installation of PostGIS (geo-spatial) extensions*: If you use RDS for serving an Arches database, you may want to review [official documentation](#) on how to add the required PostGIS extensions.
2. *Arches Allowed Hosts*: In `settings.py` (sometimes set via `settings_local.py`) you will need to add multiple items to the list of `ALLOWED_HOSTS`. Consider the following example:

```
ALLOWED_HOSTS = ["my-arches-site.org", "localhost", "127.0.0.1", "ip-10-xxx-x-x.eu-west-2.compute.internal", "10.xxx.x.x", "ip-10-xxx-x-x"]
```

In that example, “my-arches-site.org” is the public domain name. But the items “ip-10-xxx-x-x.eu-west-2.compute.internal”, “10.xxx.x.x”, and “ip-10-xxx-x-x” are all AWS internal network addresses for the EC2 instance where Arches is deployed. You may need all of these for Arches to work properly.

3. *Arches CSRF Trusted Origins*: Django 4.0, a dependency of Arches 7.5 introduced a new setting for security purposes. In the `settings.py` (sometimes set via `settings_local.py`) you will need to add multiple items to the list of `CSRF_TRUSTED_ORIGINS`. If you don’t include this, users will encounter CSRF error (403) then they attempt to login. See the [Django documentation for details](#). Note the following items (with the `https://` prefix):

```
CSRF_TRUSTED_ORIGINS = ["https://my-arches-site.org", "https://www.my-arches-site.org",]
```

Next Steps: Configuration with Apache or Nginx

Once you’ve verified that you have properly installed Arches and its dependencies on your EC2 instance, it’s time to configure Arches to work with either Apache or Nginx web servers. Apache or alternatively Nginx play an important role in security and performance. Configuring Apache or Nginx is a *necessary* aspect of deploying Arches in production. Please review [Serving Arches with Apache or Nginx](#) to learn more about production deployment of Arches.

HTTP ROUTING TABLE

/geojson

GET /geojson, 112

/history

GET /history/, 108

GET /history/{int: page number}, 108

/mobileprojects

GET /mobileprojects, 111

/o

POST /o/token, 99

/rdm

GET /rdm/concepts/{uuid:concept instance
id}, 100

/resources

GET /resources/, 102

GET /resources/{uuid:resource instance id},
103

PUT /resources/{uuid: graph
id}/{uuid:resource instance id},
105

DELETE /resources/{uuid:resource instance
id}, 107