# CollectiveAccess Documentation

*Release 1.8*

**Whirl-i-Gig**

**Dec 13, 2020**

# Introduction to CollectiveAccess

CollectiveAccess is open-source collections management and presentation software designed for museums, archives, and special collections also increasingly used by libraries, corporations and non-profits. It is designed to handle large, heterogeneous collections that have complex cataloguing requirements and require support for a variety of metadata standards and media formats. CollectiveAccess is a collaboration between Whirl-i-Gig and partner institutions in North America and Europe with projects in 5 continents. The software is freely available under the open source GNU Public License, meaning it's not only free to download and use but that users are encouraged to share and distribute code.

# CHAPTER 1

# Contents

## 1.1 What is CollectiveAccess?

### 1.1.1 Who Uses CA?

### 1.1.2 Why Should I Use It?

## 1.2 System Requirements

### 1.2.1 What is Providence?

**Providence** is the core of CollectiveAccess. It includes a data modeling framework, a database, a media handling framework capable of manipulating and converting digital images, video, audio and documents, and a web-based user interface application for cataloguing, searching and managing your collections. If you are starting out with CollectiveAccess, **Providence** is the first (and most important) component you need to install. All other CollectiveAccess components are add-ons to Providence and require a functional Providence installation.

### 1.2.2 Getting Started

Providence is a web-based application that runs on a server. Users access the server from their own computers over a network using standard web browser software. As with any web-based application, Providence is designed to be accessed via the internet, enabling collaborative cataloguing of collections by widely dispersed teams. However, you do **not** have to make your Providence installation accessible on the internet. It will function just as well on a local network with no internet connectivity, or even on a single machine with no network connectivity at all. Who gets to access your system is entirely up to you.

Before attempting an installation verify that your server meets the basic requirements for running Providence:

| Server Requirements | Notes |
|---|---|
| Operating System | Linux, Mac OS X 10.9+, or Windows (Server 2012+, Windows 7, 8 and 10 verified to work). |
| Server Memory | 4 gb of RAM minimum. If you intend to have CA handle large image files then your server should ideally have three times the size of the largest image when uncompressed. In general more memory is always better, and 8 gb of RAM is a good baseline assuming it is not cost prohibitive. |
| Data Storage | A simple formula for estimating storage requirements requires an expected number of media items to be catalogued and an average size for those media items. Once these quantities are known an estimate can be derived using some simple arithmetic: <storage required in mb> = (<# of media items> * <average storage requirements per media item in mb>) + (<# of media items> * 5mb). 5mb is estimated overhead of storing derivatives (small JPEG, TilePic pan-and-zoom version, etc.) It is recommended to double the calculated storage requirements when acquiring hardware if practical. Storage requirements for your metadata and database indices, even if your database is quite large, are usually negligible compared to the storage required for media. |
| Processor | Multiprocessor/multicore architectures are desirable for the improved scalability they provide, and well as the capability to speed the processing of uploaded media. Media processing is often CPU-bound (as opposed to database operations which are often I/O bound) and lends itself to multiprocessing. It is advisable to obtain a machine with at least 2 cores and, if possible, 4+ cores. |

## 1.2.3 Core software requirements

Providence requires three core open-source software packages be installed prior to installation. Without these packages Providence cannot run:

| Software Package | Notes |
|---|---|
| Webserver | Apache version 2.4 or NGINX 1.14 or later are recommended. |
| MYSQL Database | Versions 5.5, 5.6, 5.7 and 8.0 are supported. |
| PHP programming language | PHP version 7.0 or better is required. PHP 7.2 or later is strongly recommended. Note that the CollectiveAccess, 1.7.7 is the last version to support PHP 5.6. |

All of these should be available as pre-compiled packages for most Linux distributions and as installer packages for Windows. For Macs, Brew is a highly recommended way to get all of CA's prerequisites quickly up and running.

If setting up Apache, MySQL or PHP is daunting, you may want to consider pre-configured Apache/MySQL/PHP environments available for Windows and Macintosh such as MAMP and XAMPP. These can greatly simplify setup of CollectiveAccess and its' requirements and are useful tools for experimentation and prototyping. They are not recommended for hosting live systems, however.

## 1.2.4 Required and Suggested Software Packages By Distribution

### CentOS 7

Some packages used by CollectiveAccess are available only from 3rd party repositories. Packages recommended here are from the following repositories:

- Nux: http://li.nux.ro/download/nux/dextop/el7/x86_64/nux-dextop-release-0-5.el7.nux.noarch.rpm
- Remi: http://rpms.remirepo.net/enterprise/remi-release-7.rpm

- EPEL: https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

**Required:**

- mariadb-server [Database server]
- httpd [Web server]
- redis-server [Cache server]
- php php-mcrypt php-cli php-gd php-curl php-mysqlnd php-zip php-fileinfo php-devel php-gmagick php-opcache php-process php-xml php-mbstring php-redis [Runtime environment] (Remi, EPEL)

**Suggested:** - GraphicsMagick-devel [Image processing] - ghostscript-devel - ffmpeg-devel [Audio and video processing] (Nux) - libreoffice [Microsoft Office file processing] (EPEL) - dcraw [RAW image format support] - mediainfo [Media metadata extraction] - exiftool [Media metadata extraction] - xpdf [Media metadata extraction]

When installing a tool for media metadata extraction, you need only install one, although having multiple installed will not cause issues.

**Ubuntu 16.04**

Some packages used by CollectiveAccess are available only from 3rd party repositories. Packages recommended here are from the following repositories:

- ondrej/php: ppa:ondrej/php
- PECL: https://pecl.php.net

**Required:**

- mysql-server
- apache2
- redis-server
- php7.x libapache2-mod-php7.x php7.x-common php7.x-mbstring php7.x-xmlrpc php7.x-gd php7.x-xml php7.x-intl php7.x-mysql php7.x-cli php7.x-mcrypt php7.x-zip php7.x-curl php7.x-posix php7.x-dev php-pear php7.x-
- pecl.php.net/gmagick-2.0.5RC1 [pecl install channel://pecl.php.net/gmagick-2.0.5RC1]

**Suggested:**

- graphicsmagick libgraphicsmagick-dev [Image processing]
- ffmpeg [Audio and video processing]
- ghostscript [PDF processing]
- libreoffice [Microsoft Office file processing]
- dcraw [RAW image format support]
- mediainfo [Media metadata extraction]
- xpdf [Media metadata extraction]
- exiftool [Media metadata extraction]

### 1.2.5 Directories

If you are running Apache on Linux, the root of your CollectiveAccess installation will usually be located in **/var/www/html.**

### 1.2.6 Software requirements for media processing

Depending upon the types of media you intend to handle with CA you will also need to install various supporting software libraries and tools. None of these is absolutely required for CA to install and operate but without them specific types of media may not be supported (as noted below).

| Software Package | Media Types | Notes |
|---|---|---|
| GraphicsMagick | Images | Version 1.3.16 or better is required. GraphicsMagick is the preferred option for processing image files on all platforms and is better performing than any other option. Be sure to compile or obtain a version of GraphicsMagick with support for the formats you need. Support for some image formats is contingent upon other libraries being present on your server (eg. libTiff must be present for TIFF support]). Some less common formats, such as PSD, may require special configuration and/or compilation. |
| ImageMagick | Images | Version 6.5 or better is required. ImageMagick can handle more image formats than any other option but is significantly slower than GraphicsMagick in most situations. Be sure to compile or obtain a version of ImageMagick with support for the formats you need! Support for some image formats is contingent upon other libraries being present on your server (eg. libTiff must be present for TIFF support]). |
| libGD | Images | A simple library for processing JPEG, GIF and PNG format images, GD is a fall-back for image processing when ImageMagick is not available. This library is typically bundled with PHP so you should not need to install it separately. In some cases you may need to perform a manual install or use a package provided by your operating system provider. In addition to supporting a limited set of image formats, GD is typically slows than ImageMagick or GraphicsMagick for many operations. If at all possible install GraphicsMagick on your server. |
| ffmpeg | Audio, Video | Required if you want to handle video or audio media. Be sure to compile to support the file formats and codecs you require. |
| Ghostscript | PDF Documents | Ghostscript 8.71 or better is required to generate preview images of uploaded PDF documents. PDF uploads will still work, but without preview images, if Ghostscript is not installed. If you require color management (if you are dealing with color PDF documents you do), then you must install Ghostscript 9.0 or better. |
| dcraw | Images | Required to support upload of proprietary CameraRAW formats produced by various higher-end digital cameras. Note that that AdobeDNG format, a newer RAW format, is supported by GraphicsMagick and ImageMagick. |
| PdfToText | PDF Documents | A utility to extract text from uploaded PDF files. If present CA will use PdfToText to extract text for indexing. If PdfToText is not installed on your server CA will not be able to search the content of uploaded PDF documents. |
| PdfMiner | PDF Documents | A utility to extract text and text locations from uploaded PDF files. If present CA will use PdfMiner to extract text for indexing and locations to support highlighting of search results during PDF display. If PdfMiner is not installed on your server CA will fall back to PdfToText for indexing and highlighting of search results will be disabled. |
| MediaInfo | Images, Audio, Video, PDF Documents | A library for extraction of technical metadata from various audio and video file formats. If present CA can use MediaInfo to extract technical metadata, otherwise it will fall back to using various built-in methods such as GetID3. |
| ExifTool | Images | A library for extraction of embedded metadata from many image file formats. If present CA can use it to extract metadata for display and import. |
| WkHTMLToPDF | PDF Output | WkHTMLToPDF is an application that can perform high quality conversion of HTML code to PDF files. If present CollectiveAccess can use WkHTMLToPDF to generate PDF-format labels and reports. Version 0.12.1 is supported. Do not use version 0.12.2, which has bugs that prevent valid formatting of output. If WkHTMLToPDF is not installed CollectiveAccess will fall back to a slower |

Most users will want at a minimum GraphicsMagick and ffmpeg installed on their server, and should install other packages as needed. For image processing you need only one of the following: GraphicsMagick, ImageMagick, libGD.

### 1.2.7 PHP extensions for media processing (optional)

CA supports two different mechanisms to employ GraphicsMagick or ImageMagick. The preferred option is a PHP extensions that, when installed, provide a fast and efficient way for PHP applications such as CA to access GraphicsMagick or ImageMagick functionality. Alternatively GraphicsMagick or ImageMagick can be invoked as a command-line program directly without any PHP extension.

In general you should try to use a PHP extension rather than the command-line mechanism. The extensions provide **much** better performance. Unfortunately, the extensions have proven to be unstable in some environments and can be difficult to install on Windows systems. If you are running the PHP GMagick (for GraphicsMagick) or IMagick (for ImageMagick) extension and are seeing segmentation faults or incorrect image encoding such as blank images you should remove the extension, let the command-line mechanism take over and see if that improves things.

**Note:** GraphicsMagick version 1.3.32 and better break certain functions in the PHP GMagick extension API and cause all media processing to fail in CollectiveAccess in versions prior to 1.7.9. Upgrade to the current version of CollectiveAccess if you are seeing failed processing with later versions of GraphicsMagick from 1.3.32.

Both Gmagick and Imagick are available in the PHP PECL repository and often available as packages for various operating systems. They should be easy to install on Unix-y operating systems like Linux and Mac OS X. Installation on Windows is a waking nightmare.

### 1.2.8 Configuring PHP prior to installation

With the core software requirements installed on your server examine the newly installed PHP configuration file. A few settings may need adjustment.

Your PHP configuration file is usually named php.ini. On Linux systems the php.ini file is often in /etc/php.ini or /usr/local/lib/php.ini. If you cannot locate your php.ini file, look for its location in the output of phpinfo(), either by running the PHP command line interpreter with the -i option (eg. **php -i**) or running a PHP script that looks like this: **<?php phpinfo(); ?>** The output from phpinfo() will include the precise location of the php.ini file used to configure PHP.

Once you've found your php.ini file verify and, if necessary, change the following values:

1. *post_max_size* - sets maximum size a POST-style HTTP request can be. The default value is 8 megabytes. If you are uploading large media files (and most CollectiveAccess users are) you will need to raise this to a value larger than the largest file size you are likely to encounter.
2. *upload_max_filesize* - sets the maximum size of an uploaded file. Set this to a the same large value set for post_max_size.
3. *memory_limit* - sets the maximum amount of memory a PHP script may consume. The default is 128 megabytes which should be enough for many systems, unless you are (a) uploading large images (b) reindexing the search index of a large database or (c) importing data. Even if you have not received memory limit exceeded errors, you may want to increase this limit to 196 or 256 megabytes.
4. *display_errors* - determines whether errors are printed to the screen or not. In some installation this is set to "off" by default. While this is a good security decision for public-facing systems, it can make debugging installation problems difficult. It is therefore suggested that while installing and testing CA you set this option to "On"

### 1.2.9 Installing Providence

To install CollectiveAccess Providence perform the following steps:

1. Set up an empty MySQL database for your installation. Give the database a name and create a login for it with full read/write access. Note the login information - you'll need it later. You can use the MySQL command line or web-based tools like phpMyAdmin to create the database and login.
2. Copy the contents of the CollectiveAccess software distribution to the root of the web server instance in which your installation will run. You can obtain the latest release version from our download page. If you wish to obtain CollectiveAccess from the project's GitHub repository run the following command from the parent of the directory into which you want to install CA: `git clone https://github.com/collectiveaccess/providence.git providence` where the trailing "providence" is the name of the directory you want your installation to be in. Git will create the directory for you.
3. Copy the setup.php-dist file (in the root directory of the CA distribution) to a file named setup.php. Edit setup.php, changing the various directory paths and database login parameters to reflect your server setup.
4. Make sure the permissions on the `app/tmp`, `app/log`, `vendor/ezyang/htmlpurifier/library/HTMLPurifier/DefinitionCache` and `media` directories are such that the web server can write to them. In the next step, the web-based installer will need the access to create directories for uploaded media, and to generate cached files. In most hosted environments these permissions will already be set correctly.
5. In a web browser navigate to the web-based installer. If the URL for your installation server is `http://www.myCollectiveaccessSite.org` then the URL to the installer is `http://www.myCollectiveaccessSite.org/install`. Enter your email address and select the installation profile (a profile is a set of pre-configured values for your system) that best fits your needs. Then click on the "begin" button. If you don't see a profile suitable for your project you may want to ask on the support forum or look at our list of contributed profiles.
6. The installer will give you login information for your newly installed system when installation is complete. Be sure to note this information in a safe place!

### 1.2.10 Optional post installation tasks

#### Set up for background encoding of media

By default, CollectiveAccess will process all uploaded media immediately at time of upload. For large media files this can make the user's browser in unresponsive for an extended period of time while CA performs large and complex media conversions. If you expect to be uploading many large media files you can enable background processing of media by setting the __CA_QUEUE_ENABLED__ setting to 1 in your **setup.php** (it is off by default).

Once background processing is enabled, all media files exceeding a specific size will be queued for later processing. Small sizes will still be run "while you wait" unless you modify the media processing configuration. To actually process the images in the queue you must run the script **support/bin/caUtils process-task-queue**. This script is typically run from a **crontab** (in Unix-like operating systems, at least).

You can run the queue processing script as often as you want. Only a single instance of the script is allowed to run at any given time, so you need not worry about out-of-control queue processing scripts running simultaneously and depleting server resources. Note that the queue processing script should *always* be run under a user with write-access to the CA media directory.

### 1.2.11 What to do if something goes wrong?

**Tip:** If your CollectiveAccess installation fails, the first thing to do is examine error messages on screen or in the log (written to the app/log directory). If you receive a blank white screen odds are error messages are being suppressed in your PHP php.ini configuration file. Try changing the **display_errors** option to "On" and then attempt to reinstall.

If you are totally stumped after reviewing the error messages and logs you can find help on the online support forum. Please include a full description of your problem as well as the operating system you are running, the version of CA you are running, the text of any error messages, the output of phpinfo() and the output of the CA "Configuration Check" (available in the "Manage" menu under "System Configuration") - assuming you are able to log in. We will try our best to resolve your problems quickly.

You may also want to look at our list of OS specific *Installation* notes.

## 1.3 Backing up a CollectiveAccess installation

Three types of data need to be backed up on a regular basis:

- The database
- Digital media
- Configuration files

All of the procedures outlined in this document assume that you're using a tool that backs up files - something that copies files on your system to some other archival medium, and can restore them from that medium as needed. This could be a program such as BRU (http://www.tolisgroup.com/), AMANDA (http://www.amanda.org/) or Bacula (http://www.bacula.org/) archiving your files to digital tape or a command-line Unix program such as rsync mirroring files from your server to another server or an external hard drive.

### 1.3.1 Backing up your database

To back-up your CollectiveAccess database, you need to have MySQL "dump" it to a file and then have your back-up tool archive that file. MySQL comes with a command-line program called mysqldump that can create a complete snapshot of a database in a single file. The snapshot file will contain SQL commands to restore both the structure of the database and all of the data. A typical command-line invocation of mysqldump would look something like this:

```
mysqldump -udb_login_name -pdb_login_password database_name > /path/ to/dumpfile/my_
↪database_backup.dump
```

where the *db_login_name*, *db_login_password* and *database_name* reflect the settings on your system, and *my_database_backup.dump* is the name of the newly created file.

You can automate the execution of mysqldump by adding an invocation to your **crontab** (on Unix- like systems) or equivalent on Windows. A more featureful solution is a MySQL backup automation script such as AutoMySQLBackup which can take care of naming and compression of snapshots, and can easily handle multiple databases.

### 1.3.2 Backing up your digital media

CollectiveAccess stores all uploaded and derived digital media in a series of sub-directories under /media in the root of your installation. Simply backing up the entire contents of media is sufficient in most cases.

### 1.3.3 Backing up your configuration files

CollectiveAccess configuration files are stored in a sub-directory named conf under app (aka. **app/conf**). This entire directory should be backed up. Your **setup.php** file, located in the root of your installation and which contains some basic configuration information such as the locations of the application configuration file, should also be backed up.

### 1.3.4 Summary

For a typical CollectiveAccess installation where media and configuration files are stored in the typical (and pre-configured) locations, and assuming that you are writing database snapshots into a "dumps" directory in a location outside of the web server root, you should be backing-up, at a minimum, the following directories:

- /path/to/mysql/dumps
- /path/to/collectiveaccess/app/conf
- /path/to/collectiveaccess/media
- /path/to/collectiveaccess/setup.php

Depending upon the setup and size of your CollectiveAccess installation, server and back-up system you may elect to simply back-up the entire CollectiveAccess directory structure including the application code and supporting directories, rather than specifically selecting the directories above. This has the advantage of providing a complete ready-to-run backup and is the preferred option if it is possible. If you cannot do this, you can always download the CollectiveAccess application code at CollectiveAccess.org.

## 1.4 Installation

### 1.4.1 Installing on Linux

#### Ubuntu

#### Ubuntu 16.04LTS

Packages:

apt install -y git screen mysql-server ghostscript libgraphicsmagick-dev xpdf dcraw redis-server ffmpeg exiftool libreoffice apache2 systemctl enable apache2.service systemctl start apache2.service apt-get install -y software-properties-common add-apt-repository ppa:ondrej/php apt install -y php7.2 libapache2-mod-php7.2 php7.2-common php7.2-mbstring php7.2-xmlrpc php7.2-gd php7.2-xml php7.2-intl php7.2-mysql php7.2-cli php7.2-zip php7.2-curl php7.2-posix php7.2-dev php-pear php7.2-redis php7.2-gmagick php7.2-gmp

#### Ubuntu 18.04LTS

To come

### Red Hat Enterprise Linux/CentOS

#### RHEL/CentOS 7

Packages:

yum -y install yum-utils mariadb-server ghostscript ghostscript-devel

yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm yum -y install http://rpms.remirepo.net/enterprise/remi-release-7.rpm yum -y install GraphicsMagick-devel

yum -y install httpd yum-config-manager –enable remi-php73 yum -y install php php-mcrypt php-cli php-gd php-curl php-mysqlnd php-zip php-fileinfo php-devel php-gmagick php-opcache php-process php-xml php-mbstring php-redis redis

rpm -Uvh http://li.nux.ro/download/nux/dextop/el7/x86_64/nux-dextop-release-0-5.el7.nux.noarch.rpm yum -y install ffmpeg ffmpeg-devel

yum -y install xpdf dcraw mediainfo git screen wkhtmltopdf yum -y install mod_ssl openssl

systemctl enable mariadb systemctl start mariadb systemctl enable httpd systemctl start httpd systemctl enable redis systemctl start redis

#### RHEL/CentOS 8

yum -y install mariadb-server dnf -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm dnf -y install https://rpms.remirepo.net/enterprise/remi-release-8.rpm dnf -y install yum-utils dnf config-manager –set-enabled remi dnf -y install redis httpd mod_ssl dnf -y module install php:remi-7.3 dnf -y install git screen dnf -y install php-cli php-gd php-curl php-mysqlnd php-zip php-fileinfo php-gmagick php-opcache php-process php-xml php-mbstring php-redis redis

dnf -y install ghostscript

dnf install –nogpgcheck https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm dnf install –nogpgcheck https://download1.rpmfusion.org/free/el/rpmfusion-free-release-8.noarch.rpm https://download1.rpmfusion.org/nonfree/el/rpmfusion-nonfree-release-8.noarch.rpm dnf config-manager –enable PowerTools

dnf -y install ffmpeg

firewall-cmd –zone=public –add-service=http –permanent firewall-cmd –zone=public –add-service=https –permanent firewall-cmd –reload

systemctl enable mariadb systemctl start mariadb systemctl enable httpd systemctl start httpd

## 1.4.2 Installing on Mac OS

**Note:** Note: these instructions have been tested on MacOS 10.14 (Mojave). They may or may not work on earlier versions of MacOS.

CollectiveAccess relies on a number of open-source software packages to run, such as MySQL (database server), PHP (programming lanaguage) and Apache or nginx (web server) to name just a few. The simplest way to install these required packages on Mac OS is to use the Homebrew package manager. Homebrew can be installed by opening a Mac OS Terminal window and pasting this command:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

Once installed most required software can be installed using the *brew* command.

Mac OS 10.14 comes with the Apache web server preinstalled. It's tricky to get PHP installed by Homebrew to work with the preinstalled Apache though, so it's best to use Homebrew-managed installation. Before we install Apache with Homebrew, first shutdown the preinstalled server and disable it from starting automatically in the future using these Terminal commands:

```
sudo apachectl stop
sudo launchctl unload -w /System/Library/LaunchDaemons/org.apache.httpd.plist 2>/dev/
→null
```

Now install Apache by typing in Terminal:

```
brew install httpd
```

Next, set Apache to start itself automatically every time you reboot the Mac:

```
sudo brew services start httpd
```

You should now be able to connect to the web server on port 8080 (the default when installing with Brew) by going to the URL *http://localhost:8080* in a web browser running on the Mac. The message "It works!" should display.

**Tip:** If you want to run Apache on the standard port 80 you'll need to open the Apache configuration file located at */usr/local/etc/httpd/httpd.conf*, find the line *Listen 8080* and change it to *Listen 80*. Then restart the server with the Terminal command *sudo apachectl -k restart*

Next install PHP version 7.2 running in the Terminal:

```
brew install php@7.2
```

Then edit the Apache configuration file located at */usr/local/etc/httpd/httpd.conf*, adding the line:

```
LoadModule php7_module /usr/local/opt/php@7.2/lib/httpd/modules/libphp7.so
```

Next, look for this configuration in the Apache configuration file:

```
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
```

and replace it with this:

```
<IfModule dir_module>
    DirectoryIndex index.php index.html
</IfModule>

<FilesMatch \.php$>
    SetHandler application/x-httpd-php
</FilesMatch>
```

Restart the server with the Terminal command *sudo apachectl -k restart*. You should now have PHP enabled within your Apache web server.

In order to use the PHP on the Terminal command line (which can be handy) you'll need to add the Homebrew PHP installation directory into your command PATH. Do this by entering in the Terminal:

```
echo 'export PATH="/usr/local/opt/php@7.2/bin:$PATH"' >> ~/.bash_profile
echo 'export PATH="/usr/local/opt/php@7.2/sbin:$PATH"' >> ~/.bash_profile
```

Close the current Terminal window and open a new one. Typing *php -v* in the Terminal should return output similar to:

```
PHP 7.1.23 (cli) (built: Feb 22 2019 22:08:13) ( NTS )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.1.0, Copyright (c) 1998-2018 Zend Technologies
```

Now let's install MySQL. CollectiveAccess works with version 5.7. It is not yet compatible with version 8.0. To install version 5.7:

```
brew install mysql@5.7
```

Then add the MySQL install to your command line path with:

```
echo 'export PATH="/usr/local/opt/mysql@5.7/bin:$PATH"' >> ~/.bash_profile
```

You will need to close the Terminal window and open a new one for the path changes to take effect. Next start up MySQL, and configure it to restart automatically on reboot:

```
brew services start mysql@5.7
```

If you don't want MySQL starting up automatically every time you boot your machine you can start it up on demand using *brew services run mysql@5.7*

Next we install various packages to support processing of media: ffmpeg (audio/video), Ghostscript (PDFs), GraphicsMagick (mages), mediainfo (metadata extraction and xpdf (content extraction from PDFs):

```
brew install ffmpeg ghostscript GraphicsMagick mediainfo xpdf
```

Finally, we are ready to install the CollectiveAccess *Providence* back-end cataloguing application. The web server we installed earlier uses */usr/local/var/www* for documents by default (the "web server root" directory). We are going to place CollectiveAccess in this directory, in a subdirectory named *ca*. A URL for this directory will be http://localhost:8080/ca (assuming that you're still running on port 8080). If you're running on port 80, the URL will be http://localhost/ca.

**Tip:** You can use a different directory for the application by editing */usr/local/etc/httpd/httpd.conf*. Edit the line *DocumentRoot "/usr/local/var/www"* to point to your chosen directory.

You can download a release from https://github.com/collectiveaccess/providence/releases, or install is with Git. Using a release in somewhat simpler to install, while using Git allows you to easily update files and switch to development versions of CollectiveAccess.

To install with Git, in the Terminal change directory into the web server root directory.

```
cd /usr/local/var/www
```

Then "clone" the Providence application code from GitHub:

```
git clone https://github.com/collectiveaccess/providence.git ca
```

If you prefer to download a release, place the release ZIP or tgz file downloaded from https://github.com/collectiveaccess/providence/releases into /usr/local/var/www and uncompress it. Then rename the resulting directory (named something like *providence-1.7.8*) to *ca*.

In the Terminal change directory into the *ca* application directory and copy the *setup.php-dist* file to *setup.php*. This file contains basic configuration for Providence. The "-dist" version is simply a template. The *setup.php* copy will need to be customized for your installation:

```
cd /usr/local/var/www/ca
cp setup.php-dist setup.php
```

Edit *setup.php*, changing settings to suit. At a minimum you will need to edit the database login settings *__CA_DB_USER__*, *__CA_DB_PASSWORD__*, *__CA_DB_DATABASE__*. You may want to edit other settings, which are described in setup.php. You should also edit the *__CA_STACKTRACE_ON_EXCEPTION__* to be true. This will allow you to receive full error messages on screen if something goes wrong.

By default brew installs the MySQL database with an all-access, password-less administrative account named *root*. It's generally insecure to leave this account password-less, but in a testing environment this may not matter. If you decide to use the root account, set *__CA_DB_USER__* to "root", leave *__CA_DB_PASSWORD__* blank and set *__CA_DB_DATABASE__* to the name you'll use for your database. For this example, we'll assume the database is to be named *my_archive*.

MySQL can support multiple databases in a single installation, so the *my_archive* database must be created explicitly. Log into mysql in the Terminal using the *mysql* command (assuming you haven't set a password for the root account):

```
mysql -uroot
```

At the *mysql>* prompt enter:

```
CREATE DATABASE my_archive;
```

If you want to use a MySQL login specific to the newly created database, while still at the *mysql>* prompt enter:

```
GRANT ALL on my_archive.* to my_user@localhost identified by 'my_password';
```

where *my_user* is your preferred MySQL user name and *my_password* is your preferred password for the MySQL login. MySQL logins are specific to MySQL and have nothing to do with your server login. You can set the user name and password to whatever you want, independent of all other login credentials.

Go back to *setup.php* and enter your MySQL login credentials into the *__CA_DB_USER__*, *__CA_DB_PASSWORD__* and *__CA_DB_DATABASE__* settings.

Restart the server with the Terminal command:

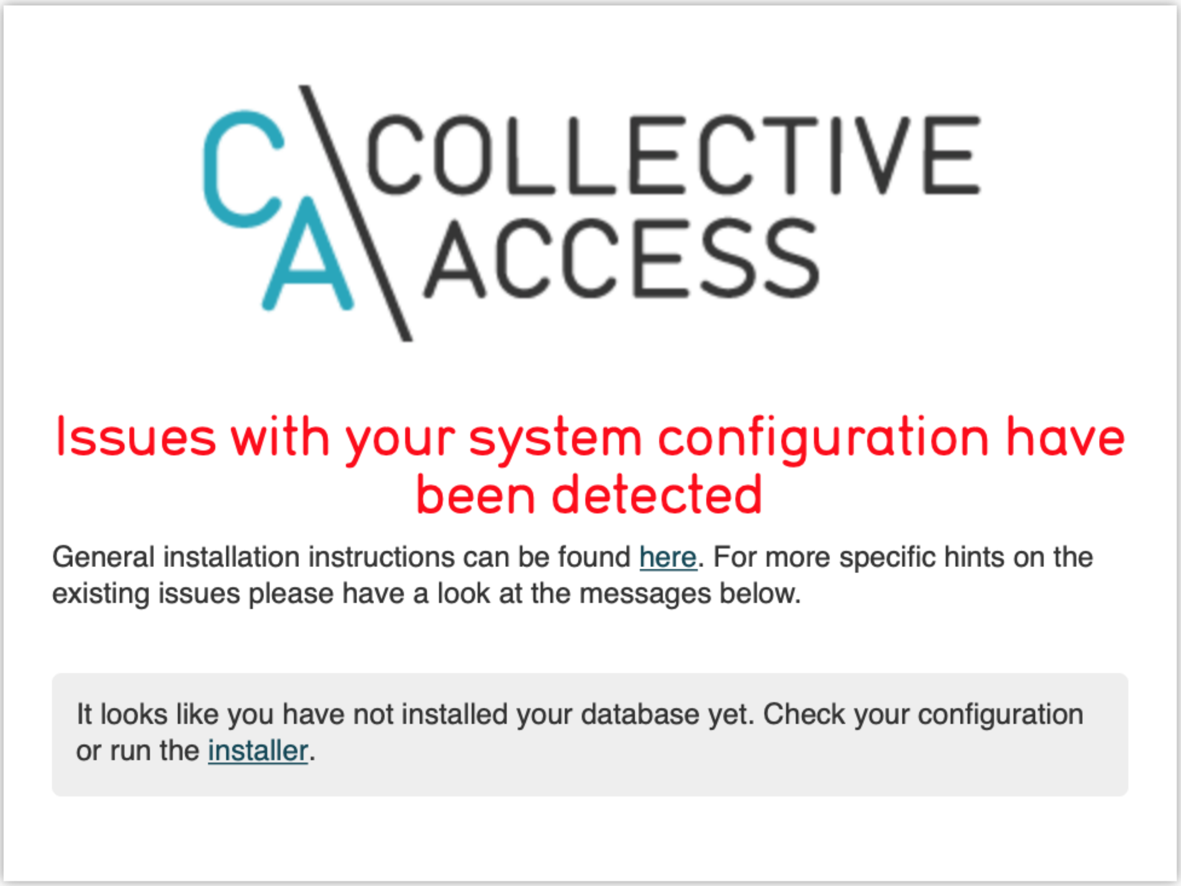
```
sudo apachectl -k restart
```

Certain directories in the installation need to be writeable by the web server, within which CA runs. On MacOS, the web server typically runs as the user *www*. Change the permissions on the *app/tmp*, *app/log*, *media* and *vendor* directories to be writeable by *www* in Terminal:

```
cd /usr/local/var/www/ca
sudo chown -R www app/tmp app/log media vendor
sudo chmod -R 755 app/tmp app/log media vendor
```

The first *sudo* command will require entry of your MacOS login password.

Navigate in a web browser to http://localhost:8080/ca (adjusting the port to whatever you have configured, if necessary). You should see:

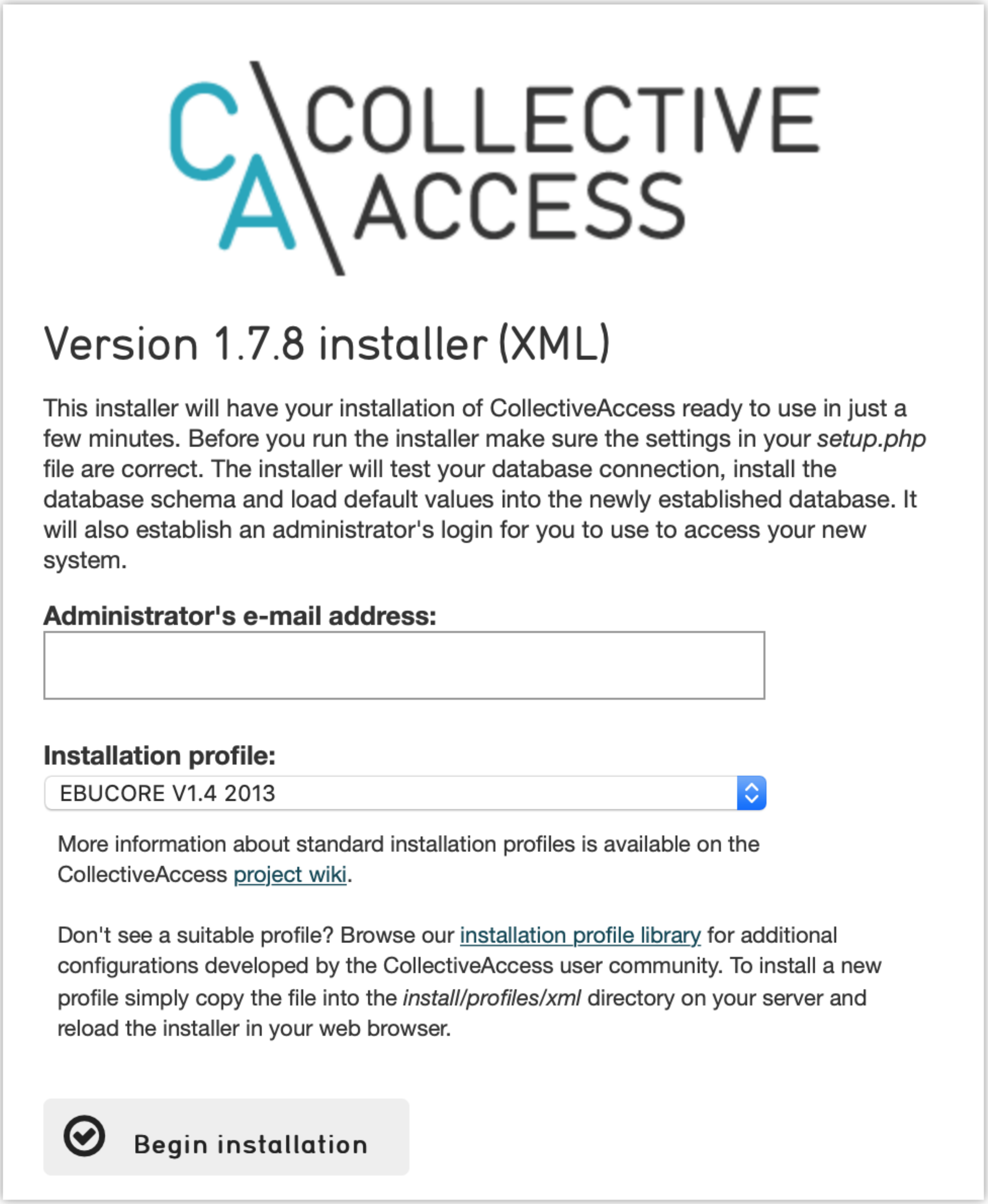


Issues with your system configuration have been detected

General installation instructions can be found here. For more specific hints on the existing issues please have a look at the messages below.

It looks like you have not installed your database yet. Check your configuration or run the installer.

Click on the *installer* link and you should see:



Version 1.7.8 installer (XML)

This installer will have your installation of CollectiveAccess ready to use in just a few minutes. Before you run the installer make sure the settings in your *setup.php* file are correct. The installer will test your database connection, install the database schema and load default values into the newly established database. It will also establish an administrator's login for you to use to access your new system.

**Administrator's e-mail address:**

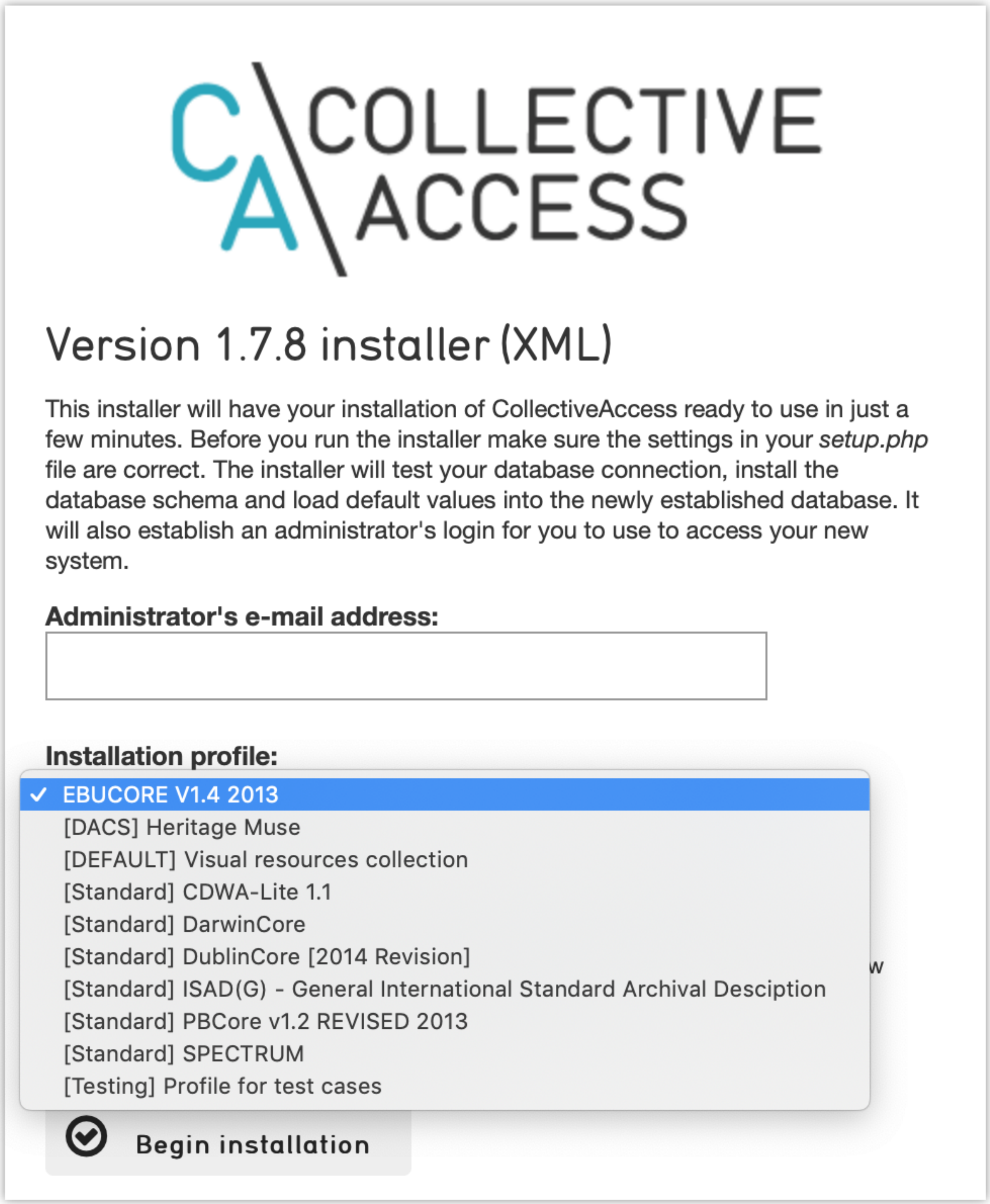
**Installation profile:**

EBUCORE V1.4 2013

More information about standard installation profiles is available on the CollectiveAccess project wiki.

Don't see a suitable profile? Browse our installation profile library for additional configurations developed by the CollectiveAccess user community. To install a new profile simply copy the file into the *install/profiles/xml* directory on your server and reload the installer in your web browser.

Select a profile, enter your email address and click on *Begin installation*. A profile is a preset template with record types, fields and other cataloguing settings that the installer uses to define a new working system. The standard profiles Providence ships with include implementations of widely used standards:


Version 1.7.8 installer (XML)

This installer will have your installation of CollectiveAccess ready to use in just a few minutes. Before you run the installer make sure the settings in your *setup.php* file are correct. The installer will test your database connection, install the database schema and load default values into the newly established database. It will also establish an administrator's login for you to use to access your new system.

**Administrator's e-mail address:**

**Installation profile:**

- ✓ EBUCORE V1.4 2013
- [DACS] Heritage Muse
- [DEFAULT] Visual resources collection
- [Standard] CDWA-Lite 1.1
- [Standard] DarwinCore
- [Standard] DublinCore [2014 Revision]
- [Standard] ISAD(G) - General International Standard Archival Desciption
- [Standard] PBCore v1.2 REVISED 2013
- [Standard] SPECTRUM
- [Testing] Profile for test cases

You can add your own profiles, or use profiles from other users by dropping profile files in the */usr/local/var/www/ca/install/profiles/xml* directory.

If you want to experiment with different profiles you may wish to set the *__CA_ALLOW_INSTALLER_TO_OVERWRITE_EXISTING_INSTALLS__* option in setup.php. By default the installer will refuse to install over an existing installation. With *__CA_ALLOW_INSTALLER_TO_OVERWRITE_EXISTING_INSTALLS__* set the installer will include an option to overwrite existing data. In a real system this is **extremely** dangerous – any one with access to the installer can delete the entire system – but is very handy for testing and evaluation.

### 1.4.3 Installing on Windows

Installation of media handling libraries and delegates such as ffmpeg can be problematic because it is more difficult to build software from source on Windows. See Compiling_ffmpeg for information how to install ffmpeg for Windows.

The format of the */app/conf/External_Applications.conf* file is different in Windows installations. For example, the correct format for the entry describing the ghostscript application is

`ghostscript_app = E:/prog/gs/gswin32c.exe`

in this case, ghostscript is installed on disk E: in the subdirectory *prog/gs*. The application is the non-windows version of ghostscript.

The *app/helpers/mediaPluginHelpers.php* file must also be updated to function properly in Windows. The entry for ghostscript must be changed from

`exec($ps_path_to_ghostscript." -v 2> /dev/null", $va_output, $vn_return);`

to

`exec($ps_path_to_ghostscript." -v 2> /$null", $va_output, $vn_return);`

Similarly, all other media helper functions to detect the other processors you have installed for CA to used must be updated to change */dev/null* to `/$null`.

Other places that have */dev/null* include

TilePicParser in *applibcoreparsers* CoreImage.php in *applibcorePluginsMedia ImageMagick.php* in *applibcorePluginsMedia PDFWand.php* in *applibcorePluginsMedia*

references to /dev/null must be changed to `/\$null` in order for the plugin to work correctly. This is particularly important if you are using ImageMagic. Both ImageMagick.php and TilePicParser.php must be changed for this process to work.

The *external_applications* line for ImageMagick might be

`imagemagick_path = E:/Prog/ImageMagick`

depending on where you have installed ImageMagick. Both the static and dynamic versions of ImageMagick seem to work well.

The ImageMagick process is very slow and libGD is preferred for speed, but it requires much more memory. If you are using it locally where you have control over the memory size, up the memory limit entry of php.ini to

memory_limit = 512M

This will allow must photographs to be handled properly without the tilepic function running out of memory.

## 1.5 Setup.php

In the main directory of your Providence install, there is a file called *setup.php.dist*. Make a copy of this file and rename it *setup.php*. For your CollectiveAccess system to work, you MUST add values for your **database server hostname, user name, password, database, and administrative e-email**. You also set the site's timezone in setup.php. Most other settings can be left alone.

### 1.5.1 Database server host name

This is often set to 'localhost'.

```
if (!defined("__CA_DB_HOST__")) {
        define("__CA_DB_HOST__", 'localhost');
}
```

### 1.5.2 Database login user name

```
if (!defined("__CA_DB_USER__")) {
        define("__CA_DB_USER__", 'your_username_here');
}
```

### 1.5.3 Database login password

```
if (!defined("__CA_DB_PASSWORD__")) {
        define("__CA_DB_PASSWORD__", 'your_password_here');
}
```

### 1.5.4 Database name

```
if (!defined("__CA_DB_DATABASE__")) {
        define("__CA_DB_DATABASE__", 'your_databasename_here');
}
```

### 1.5.5 System Name

This value will be used on emails, on the login screen, in browser window titles, etc.

```
if (!defined("__CA_APP_DISPLAY_NAME__")) {
        define("__CA_APP_DISPLAY_NAME__", "insert_name_here");
}
```

### 1.5.6 Administrative Email

An e-mail must be set up at this stage to send error reports for system configuration issues.

```
if (!defined("__CA_ADMIN_EMAIL__")) {
        define("__CA_ADMIN_EMAIL__", 'example@info.com');
}
```

### 1.5.7 Outgoing email

For CollectiveAccess to be able to send email notifications __CA_SMTP_SERVER__ and __CA_SMTP_PORT__ must be set. If your outgoing (SMTP) mail server requires you to authenticate, configure your login and connection details in __CA_SMTP_AUTH__, __CA_SMTP_USER__, __CA_SMTP_PASSWORD__ and __CA_SMTP_SSL__

```
__CA_SMTP_AUTH__ = authentication method for outgoing mail connection (set to PLAIN,␣
↪LOGIN or CRAM-MD5; leave blank if no authentication is used.)
__CA_SMTP_SSL__ = SSL method to use for outgoing mail connection (set to SSL or TLS;␣
↪leave blank if not authentication is used.)
```

### 1.5.8 Timezone Setting

Set your preferred time zone here. The default is to use US Eastern Standard Time. A list of valid time zone settings is available at http://us3.php.net/manual/en/timezones.php.

**Note:** When importing data, you should switch to value 'UTC' *before* import, or else dates may import incorrectly.

```
date_default_timezone_set('America/New_York');
```

### 1.5.9 Background Processing

The task queue allows users to push potentially long running processes, such as processing of large video and image files into the background and continue working. Set this to a non-zero value if you want to use the task queue. Be sure to configure the task queue processing script to run (usually via CRON) if you set this option.

```
if (!defined("__CA_QUEUE_ENABLED__")) {
        define("__CA_QUEUE_ENABLED__", 0);
}
```

### 1.5.10 Default Locale

The default locale is used in situations where no locale is specifically set by the user, prior to login or prior to setting your preferred locale in user preferences for the first time. You should set this to the locale in which your users generally work.

**Note:** Whatever locale you set here *MUST* be present in your system locale list. The default value is US/English, which exists in most configurations.

```
if (!defined("__CA_DEFAULT_LOCALE__")) {
        define("__CA_DEFAULT_LOCALE__", "en_US");
}
```

### 1.5.11 Clean URLs

If the Apache mod_rewrite module is available on your server you may set this to have Providence use "clean" urls – urls with the index.php handler omitted. Only set this if your web server includes mod_rewrite and it is enabled using the provided .htaccess file.

```
define("__CA_USE_CLEAN_URLS__", 0);
```

### 1.5.12 App Names for Multiple CollectiveAccess Systems

If you are running more than one instance of CollectiveAccess on the same server make sure each instance has its own unique __CA_APP_NAME__ setting. __CA_APP_NAME__ must include letters, numbers and underscores only - no spaces or punctuation!

```
if (!defined("__CA_APP_NAME__")) {
        define("__CA_APP_NAME__", "your_name_here");
}
```

### 1.5.13 Google Maps Key

Add your Google Maps key to use for mapping and geocoding feature (optional).

```
if (!defined("__CA_GOOGLE_MAPS_KEY__")) {
        define("__CA_GOOGLE_MAPS_KEY__", "");
}
```

### 1.5.14 Caching

The default file-based caching should work acceptably in many setups. Alternate schema may be used, including redis, sqlite, memcached or php APC. All require additional software be present on your server, and in general all will provide better performance than file-based caching.

Options are: 'file', 'memcached', 'redis', 'apc' and 'sqlite'. Memcached, redis and apc require PHP extensions that are not part of the standard CollectiveAccess configuration check. If you do configure them here and your PHP installation doesn't have the required extension you may see critical errors. sqlite requires the PHP PDO extension and a working install of sqlite. This is not guaranteed to be present on your server, but often is.

```
if (!defined('__CA_CACHE_BACKEND__')) {
        define('__CA_CACHE_BACKEND__', 'file');
}
```

Options for the caching back-ends you may wish to set include:

```
__CA_CACHE_FILEPATH__ = Path to on on disk location for storage of cached data
__CA_CACHE_TTL__ = Cached data time-to-live (in seconds)
__CA_MEMCACHED_HOST__ = Hostname of memcached server
__CA_MEMCACHED_PORT__ = Port of memcached server
__CA_REDIS_HOST__ = Hostname of redis server
__CA_REDIS_PORT__ = Port of redis server
__CA_REDIS_DB__ = redis database index (typically a number between 0 and 15)
```

### 1.5.15 Overwrite Existing Installation

Overwriting an existing installation can be useful while a site is in development. Overwriting will completely destroy the database and anything in it, allowing you to pick a new installation profile and start over. **This option should be set back to false before delivering to a client.**

```
# Note that in overwriting your database you will destroy *all* data in the database
# including any non-CollectiveAccess tables. Use this option at your own risk!
if (!defined('__CA_ALLOW_INSTALLER_TO_OVERWRITE_EXISTING_INSTALLS__')) {
        define('__CA_ALLOW_INSTALLER_TO_OVERWRITE_EXISTING_INSTALLS__', false);
}
```

### 1.5.16 Application Exception Error Messaging

Set to display detailed error information on-screen whenever an application exception occurs. This can be helpful for developers in situtations where detailed exception messages are useful but full debugging output is not required. **For production use you should set this to false.** Note that exceptions are always logged to the application log in app/log, regardless of what is set here.

```
if (!defined('__CA_STACKTRACE_ON_EXCEPTION__')) {
        define('__CA_STACKTRACE_ON_EXCEPTION__', false);
}

require(__DIR__."/app/helpers/post-setup.php");
```

## 1.6 Introduction to Data in CollectiveAccess

### 1.6.1 Primary Tables and Records

CollectiveAccess is structured around several primary tables of metadata elements, such as Objects, Entities, Collections, and more. Each primary table has intrinsic bundles and its own set of preferred and non-preferred labels bundles.

Read more: *Primary Tables*

### 1.6.2 Bundles

*Bundle* is a high-level term for the various structures in CollectiveAccess used to store catalogued content. There are four distinct types of bundles, each with their own unique set of characteristics and uses: *labels*, *intrinsics*, *metadata elements* and *relationships*. Records are simply assemblages of various bundles, chosen to meet specific representational requirements.

#### Labels

Labels are used to store names or titles for records. Labels come in two varieties: preferred and non-preferred. Each record has one, and only one, preferred label that represents the record's current "name", and is used as the default display title.

Records may have any number of non-preferred labels. Non-preferred labels are typically used to record alternative names/titles, which may be used in searches and optionally displayed.

Both preferred and non-preferred labels are always available for all records in CollectiveAccess. No special configuration is required. Note that (with some exceptions) every record in CollectiveAccess is *required* to have a preferred label. Configuration options may be set to manadate uniqueness of labels with a system, to distinguish between different types of non-preferred labels (a requirement of some knowledge representation standard) and more.

Read more: *Labels*.

#### Intrinsics

Intrinsics are integral fields present in all CollectiveAccess systems. As with labels, intrinsics are always available and do not require special configuration. Intrinsics are simple, non-repeating values that typically exist to support specific functionality or, less often, for historical reasons. They cannot be removed from CollectiveAccess, but in most cases can be hidden if not needed.

Commonly used intrinsics include *idno* (record identifier), *type_id* (record type), *access* (public web site visibility) and *status* (record workflow status). Descriptions of all available intrinsic fields may be found in the *primary table* documentation.

#### Metadata elements

Metadata elements are configurable data fields bound to the various *records* in your data schema. Metadata element are able to accept a rich and varied range of data types, can repeat, can support multilingual values, and may be composed into complex, multi-value fields using container elements. The bulk of the data schema for a typical system will be implemented using metadata elements to build installation-specific data structures.

Read more: *Metadata Elements*

#### Relationships

Relationships are bi-directional links between pairs of records. They may be created between records in any *primary table* without restriction. All relationships include references to *relationship types* – configurable specifiers that distinguish different kinds of relationships that may occur. Relationship types between object and entity records might include, for example, “creator”, “donor” and “subject”.

Any number of relationships can be created between a pair of records, and each relationship can optionally incorporate additional metadata elements. Relationships also support a handful of intrinsics, but do not take labels.

Read more: *Relationships*

### 1.6.3 Installation Profiles

Installation profiles are the XML documents that create your data model and set up your database. Every CollectiveAccess instance must have an installation profile. Many options are pre-loaded, but typically you need to customize one for your needs.

Read more: *Installation Profiles*

## 1.7 Profiles

### 1.7.1 Metadata Standards

Standards-compliant profiles available “out-of-the-box”. You can always configure others yourself if you are willing to create an installation profile):

| Standard | Description | Support | Profile |
|---|---|---|---|
| Dublin Core | The "least common denominator" format suitable for those experimenting with cataloguing strategies, or with simple cataloguing requirements. | Dublin Core is supported as a configuration profile for item-level (object) cataloguing. The profile set up includes fields as per the Simple Dublin Core specification with a few extensions (ex. for uploaded media). Overhauled in 2014. | Dublin Core profile on Github |
| DarwinCore | The Darwin Core standard was originally conceived to facilitate the discovery, retrieval, and integration of information about modern biological specimens, their spatiotemporal occurrence, and their supporting evidence housed in collections (physical or digital). The Darwin Core today is broader in scope and more versatile. It is meant to provide a stable standard reference for sharing information on biological diversity. As a glossary of terms, the Darwin Core is meant to provide stable semantic definitions with the goal of being maximally reusable in a variety of contexts. | | DarwinCore profile on Github |
| EBU Core | Based on Dublin Core, EBUCore is a minimum list of attributes to describe audio and video resources for a wide range of broadcasting applications including for archives, exchange and publication. | Available as a configuration profile compliant with the EBU Core Version 1.4 specification. | EBU Core profile on Github |
| PB-Core | From PBCore.org:The PBCore (Public Broadcasting Metadata Dictionary) was created by the public broadcasting community in the United States of America for use by public broadcasters and related communities. Like EBUCore, the PBCore metadata specification is built on the foundation of Dublin Core, emphasizing the description of audio and video resources in production, archival, and broadcasting environments. | PBCore is supported as a configuration profile for item-level (instantiation/intellectual content) cataloguing, and as an export target. | PB-Core profile on Github |
| CDWA-Lite / CCO | From the CDWA web site: CDWA Lite is an XML schema to describe core records for works of art and material culture based on the Categories for the Description of Works of Art (CDWA) and Cataloging Cultural Objects: A Guide to Describing Cultural Works and Their Images (CCO). CA can be configured to implement the format of the XML schema in its relational database. (For those keeping score: CA does not store CDWA data in an XML format, but will be able to export data in such a format in the first version of the data export feature). | CDWA is supported as a configuration profile for item-level (object) and authority (entity, places, collections, etc.) cataloguing. An option to configure for the CDWA subset defined by Cataloguing Cultural Objects (CCO) is also provided. Will also be supported as a data export target format. This support is not yet fully implemented. | CDWA profile on Github |
| EAD | EAD is widely used in the United States as a data standard for finding aids produced by archives, libraries, museums, and manuscript repositories. | EAD is supported as a data export target format. | |
| DACS | DACS is widely used in the United States as a data content standard for finding aids produced by archives, libraries, museums, and manuscript repositories. | DACS is supported as a configuration profile for all archival levels of description. | DACS profile on Github |
| ISAD(G) | ISAD(G) is widely used data content standard for finding aids produced by archives, libraries, museums, and manuscript repositories. | ISAD(G) is supported as a configuration profile for all archival levels of description. | ISAD(G) profile on Github |
| | | | |
| VRA Core | VRA VRA Core 4.0 is a data standard for the cultural heritage community that was developed by the Visual Resources Association's Data Standards Committee. The element set provides | VRA Core 4.0 is supported as a configuration profile for item-level (object) and collection-level catalogu- | VRA Core pro- |

### 1.7.2 Configuration Library

Listed below are a selection of user-contributed installation profiles. Some are based on standards and some are completely custom; all were designed to meet the functional requirements of real-world projects. Even if none of the profiles in the library can be used "out-of-the-box" for your project, some may provide development ideas and useful points of departure for your own profile. Before attempting to decipher the profiles presented here be sure to familiarize yourself with the profile syntax, as described in the Building System Installation Profiles manual.

Note that the profiles listed here are not intended as exemplars of "good design." Many employ sub-optimal metadata and user interface structures in order to accommodate various legacy and project-specific requirements. They are presented merely as examples of how previous users have approached systems design for their particular discipline. Your mileage may vary.

To use a profile listed here with your copy of CollectiveAccess download the desired profile and copy it into the profiles/xml directory of the installer (Eg. /install/profiles/xml). After reloading the installer start page the newly installed profile should appear in the profiles drop-down menu.

#### Libraries, Museum, and Archives

| Project | Description |
|---|---|
| The Sterling Morton Library | A mixed materials system supporting archival processing, library materials, and special collections cataloging. |
| Mattress Factory | A profile supporting the documentation of site-specfic and installation-based artwork. |
| Vancouver Holocaust Education Centre | A profile for archives, library, and museum material with support for finding aids and MARC records. |
| Vancouver Maritime Museum | A mixed material maritime collection with integrated archival content. |

#### Consortiums

| Project | Description |
|---|---|
| Connecticut League Of History Organization | Collections portal for multi-institution collections aggregator. Supports museum collections cataloging as well as formal archives processing. |
| Novamuse | Collections portal for the Association of Nova Scotia Museums |

#### Institutional archives

| Project | Description |
|---|---|
| New Museum | Institutional archive for the digital preservation of program documentation for an art museum including exhibitions, public programs, education initiatives, and publications. |
| New School | A mixed material digital archive including 2D, 3D and 4D special collections content. |
| New York Society Library | This digital humanities configuration supports cataloging for data visualizations as well as special collections items. |
| Girl Scouts of the USA | An institution archive profile configuration with support for collections cataloging of photographs, documents, costumes and other archival items such as rare publications. |

### Performing Arts

| Project | Description |
|---|---|
| Brooklyn Academy of Music | A digital archive for a performing arts organization with support for documenting 150 years of performance history by Eras, Season, Works, Productions, and Special Events. Supports cataloging of photographs, moving images, documents, ephemera, promotional items, and memorabilia. |
| Jacob's Pillow Dance | A digital archive supporting the performance and production history of a renowned dance festival. |
| Roundabout Theatre Company | A digital archive for a theatre company supporting the cataloging of documents, ephemera, merchandise, costumes, props, as well as the documentation of stage productions, play readings, galas, opening nights, and special events. |

### Art galleries

| Project | Description |
|---|---|
| University of Pittsburgh Art Gallery | Supports general collections management for a university art gallery. |
| Kentler International Drawing Space | A simple collections management system for an art gallery. |

### Moving Images

| Project | Description |
|---|---|
| Smithsonian Channel | A video production and broadcast archive that support the description of moving images and supporting materials. |
| Academy of Motion Picture Arts & Sciences | A development platform for preservation workflows for digital film. |

### Paleontology

| Project | Description |
|---|---|
| iDigPaleo | The iDigPaleo (Fossil Insect Collaborative) profile makes available all the major collections of fossil insect specimens in the United States by creating electronic specimen records consisting of digital images and associated collection data. |

## 1.7.3 XML Schema

### Introduction

This manual describes how to build an installation profile using the XML-based schema, or modify an existing one. Each profile defines a number of aspects required in a working cataloguing system, including:

- Metadata Elements and Attributes. Each element definition may include the types of data an element can accept, constraints on input, how and if it repeats, where it may be used, descriptive "help" text and more. With a few exceptions there are no hardcoded fields in CollectiveAccess: you define what you need.
- Relationship types. Various types of catalogued items (objects, people, places, etc.) can be linked to each other with qualified relationships. The range of valid qualifiers for a given use-case is defined in the profile. As with metadata elements, you define only what you need for your project. You don't have to keep irrelevant options around just because the developers thought they were a good idea.
- User interfaces. There are no hardcoded editing user interfaces in CollectiveAccess. You can create as many editing interfaces as you need, each with its own unique arrangement of screen and field layouts. Each interface need only include the fields you need to edit, enabling the creation of use-specific editors for selected users and tasks. You can define user interfaces at any time using the CollectiveAccess online editor, but it is usually convenient (and highly recommended) to define at least a basic set of interfaces in a profile.
- List and Vocabulary Management. Lists are used extensively in CollectiveAccess as controlled vocabularies for cataloguing, as value sets for metadata elements, and as system lists defining the allowed values for certain application functions, such as workflow statuses and object types. While you can create new lists at any time using the web-based list and vocabulary editor, it is usually more convenient to create you basic lists in a profile. In addition, system lists, which are required for proper application function, and lists used by metadata elements defined in the profile, must be defined in the profile.

- Locales. The list of languages and cultures available for cataloguing purposes (e.g., the languages you'll be cataloguing in - not the languages or cultures used for descriptive metadata) can be defined in the profile. You can translate help text and descriptive titles for all of the lists, metadata elements and user interfaces into any language defined in the profile as a locale.

As you can see, almost every aspect of the cataloguing tool set can be customized in a profile. It may look terribly complicated, but there's no need to worry; it's really not that difficult. And for almost every type of project there are plenty of pre-built profiles available to use as a starting point.

This manual should be enough to allow you to create your own profile. Topics covered in the next sections include:

- A full explanation of the various sections and components that comprise a profile
- Modifying an existing profile and building profiles from scratch
- Installing and testing your profile

### CollectiveAccess Basics

To work effectively with profiles it is critical that you understand the fundamental structures in the CollectiveAccess database. While CollectiveAccess provides great flexibility in terms of the specifics of your data model - you define your own fields, relationships and constraints - the general structure is fixed. CollectiveAccess defines fourteen types of "items" that model the world that your collection exists in - these are referred to as the Primary Types.

Installation profiles used with CollectiveAccess version 1.0 or later are written in XML. However, the structure of the document is very similar to that used in older profiles written in the previous syntax. As with the old syntax:

- 1 When used as a value, indicates "True" or "Yes" - will allow whatever action.
- 0 When used as a value, indicates "False" or "No" - will deny whatever action.

### About Profiles

Installation profiles are "canned" configurations applied at installation time to create a working CollectiveAccess system. A profile defines all of the metadata elements, lists, locales and relationship types used by the system, and specifies how editing user interfaces for various items should operate. Because the values defined in a profile touch almost every aspect of a system, profiles can only be applied at installation time with the end result being an empty system conforming to the profile. You cannot use a profile to tweak an existing system.

Since CollectiveAccess also provides web-based tools for configuration of the same values as profiles, you may be wondering why one would choose to define a profile, an altogether more arcane approach, over using visual tools. There are several reasons:

- Profiles allow you to fully configure an arbitrarily complex system in one go at installation time. The web-based tools have no provision for batch processing. You must add each metadata element, locale, relationship type, UI specification and list item one-at-a-time. For a setup of typical complexity using the web-based tools is an exercise in extreme tedium.
- Profiles allow you to easily and repeat-ably install the same system setup across several installations. They also make it easy to share configurations with other users. The Configuration Library on the CollectiveAccess web site is a venue for sharing profiles you create with the wider CollectiveAccess user community.
- Profiles provide a convenient base for extension of basic setups for specific uses. For example, several profiles implementing popular metadata standards are included in the CollectiveAccess application package. While these profiles can be used as-is, few serious users could use them that way. Rather they are valuable as starting points to customized systems based upon a given standard. All one needs to do is create a new profile that extends the chosen standard with one's own required modifications.

### Types of Profiles in CollectiveAccess

Installation profiles for CollectiveAccess are developed for specific projects or for conformance with a metadata standard. Profiles built for projects and collections are custom tailored to meet unique requirements for that particular institution. These profiles maybe useful if you have a similar collection or project. Some examples of project-specific installation profiles include:

- Coney Island History Project, Brooklyn, NY, USA, (Historical Society Archive)
- New Museum of Contemporary Art, New York, NY, USA, (Multimedia Digital Archive)
- New School, New York, NY, USA, (Special Collections Archival Collection Management System)

Metadata standards based installation profiles create a generic cataloging interface that complies with established archival, museum, and library structure standards. These profiles offer a high level of sustainability and interoperability, but lack the custom features found in the project-specific profiles. Some examples of standards-based installation profiles include:

- Dublin Core
- PBCore
- DarwinCore
- VRACore
- MARC

Whether you choose to use a project specific profile or a standards based profile, all can be modified. You can even create your own unique profile from scratch if need be. All profiles share the same components. The following sections describe these components.

### Parts of a Profile

Installation Profiles consist of an opening declaration followed by five sections: Locale Definitions, List Definitions, Metadata Element Set (Attribute) Definitions, User Interface Definitions, and Relationship Types. Two additional and optional sections include Displays and Logins. Each of these performs a specific function within the software, and works interdependently within the profile. It does not matter in which order you define the main sections. For the optional sections Displays must come before Logins.

### Profile Declaration

Every profile begins with its declaration that sets its name, description, and other important information. The profile declaration for DublinCore looks like this:

```
<profile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"␣
↪xsi:noNamespaceSchemaLocation="profile.xsd" useForConfiguration="1"
base="base" infoUrl="http://providence.collectiveaccess.org/wiki/
↪DublinCoreInstallationProfile">
<profileName>[Standard] DublinCore</profileName>
<profileDescription>Use this profile if you want a system that is compliant␣
↪with simple DublinCore</profileDescription>
```

The "profile.xsd" link allows a profile to inherit settings from another profile. This makes it possible to define settings shared across several profiles in a single, more easily maintainable, file. To have a profile inherit from another, set the "NamespaceSchemaLocation" to your .xsd base file.

### Locale Definitions

Locale definitions specify which languages can be used for catalogued content. Any language can be coded into a profile by including its locale code, which is a combination of an ISO-639 country code and an ISO-3166-1 language code (see http://wiki.collectiveaccess.org/index.php?title=Locales for more information). Note that any locale code can be used, without restriction, for cataloguing of content. However, user interface translations (both in the CollectiveAccess application and within your profile) are limited to those locales for which application translation files have been produced. See http://wiki.collectiveaccess.org/index.php?title=Creating_a_Translalation for a current list of application translations.

Coded for English:

```
<locales>
   <locale lang="en" country="US">English</locale>
</locales>
```

Coded for English and German:

```
<locales>
   <locale lang="en" country="US">English</locale>
   <locale lang="de" country="DE">Deutsch</locale>
</locales>
```

Let's take a closer look at the above code. The parent tag states the part of the profile and the child elements define the system attributes for that part. In this case we are defining the locales, or languages, through the language and country declarations.

### List Definitions

List definitions allow you to create three types of lists: lists that define specific elements of the cataloging interface ("system lists"), lists that define set values to control content and lists that defined controlled vocabularies that can be used for descriptive cataloguing.

### System (Structure) Lists

In CollectiveAccess, you can select a cataloging interface (See Figure 1) based on what type of object, entity, collection or other item you are cataloging. Various system lists define these types.

INSERT FIGURE 1

For CollectiveAccess to function properly, 33 types of System Lists need to be present and defined for the Primary Types - objects, object events, lots, lot events, entities, places, occurrences, collections, storage locations, list items, object representations, object representation annotations and sets. All of the lists may be hierarchical, although most tend to be single-level in most cases.

### Control (User) Lists

Control lists allow you to create a field with restricted options in a cataloging record. Lists can be rendered as drop-down menus, radio button, checklists and more. For additional information see the Attribute settings: List page. Let's look closely at a control list for formats in Dublin Core:

```
<list code="dc_format" hierarchical="0" system="0" vocabulary="0">
      <labels>
        <label locale="en_US">
          <name>Dublin Core Format</name>
        </label>
      </labels>
      <items>
        <item idno="application" enabled="1" default="1">
          <labels>
            <label locale="en_US" preferred="1">
              <name_singular>Application</name_singular>
              <name_plural>Applications</name_plural>
            </label>
          </labels>
        </item>
        <item idno="audio" enabled="1" default="0">
          <labels>
            <label locale="en_US" preferred="1">
              <name_singular>Audio</name_singular>
              <name_plural>Audio</name_plural>
            </label>
          </labels>
        </item>
        <item idno="example" enabled="1" default="0">
          <labels>
            <label locale="en_US" preferred="1">
              <name_singular>Example</name_singular>
              <name_plural>Examples</name_plural>
            </label>
          </labels>
        </item>
      </items>
    </list>
```

Dublin Core actually has eight format types, but for the sake of this example, we've limited the code to just three. Now let's break it down:

- <list code> - "dc_format" is the unique code name for this list. Can not match any other list codes in the profile and can not contain spaces.
- <labels> - lets you give the list a name in defined locales, in this case English with the human readable name Dublin Core Format. If you were writing the profile to support more than one locale you would add labels for each locale. (Don't worry if you don't define labels for all allowed locales. CollectiveAccess will fall back to other languages if a label is not available in the user's language).
- hierarchical="0" - controls how data will display and be used in the editing user interface. If hierarchical is set to 1, then the list can be a multi-level hierarchy.
- system="0" - determines if the list is one of the 33 System Lists.
- vocabulary="0" - controls if the list is treated as a controlled vocabulary included in vocabulary term searches.
- item idno="application" - the unique code name for this list item
- application, audio & example - are items in the list. Note that the key for each item must be unique within the list.
- enabled, default, & labels - define how items in the list will display. If enabled is set to 0, then the item will display but not be selectable. If default is set to 1, then the item will be the default selection in the list. Be sure

to set only one item per list as default. Labels set the display label for the item. As with list preferred labels, you can define one preferred label per locale.

Once control lists are defined, they can be used in Metadata Element Set (Attribute) Definitions to catalog interface fields.

### Metadata Element Set (Attribute) Definitions

Metadata element set definitions (element sets) are templates for the various data entry units in the cataloging interface. Element sets can define something as simple as a text field or as complicated as a repeating multiline form with text fields, date fields, measurements, drop-down lists and more. Thus the "sets" moniker - a single data entry unit can be composed of any number of basic attribute types (see Attribute_Types for a full list).

Before we go further, some terminology should be defined. A metadata element set is a collection of metadata elements that defines a single editable unit of metadata. Each element is a single value of some specific type - some text, a number, a date, a measurement, a point on a map, etc. Thus, an element set is a collection of values.

An element set does not represent data. Rather, it defines the structure of data you may create during cataloguing. An attribute is data structured according to an element set. Thus, you are not creating element sets when cataloguing. Rather, you are creating attributes patterned after some element set.

Element sets are highly configurable and key to creating custom systems. While standards are in general a very good thing, our experiences working with partner institutions have shown that successful systems need to be flexible and extensible. Every collection is different and just about every cataloging project has at least a few unique requirements. A configurable system permits strict adherence to standards or customization based on the needs of the project. While this may seem a little confusing at first, it actually gives you much more control of exactly what kind of data you would like to capture and how you would like it to display.

No specific element sets are required. You need only specify those that you need for your system; however there are a number of components that comprise an element set definition.

A basic element set looks like this:

```
<metadataElement code="description" datatype="Text">
      <labels>
        <label locale="en_US">
          <name>Description</name>
          <description>An account of the resource.</description>
        </label>
      </labels>
      <documentationUrl>http://dublincore.org/documents/dcmi-terms/#terms-
↪description</documentationUrl>
      <settings>
        <setting name="usewysiwygeditor">1</setting>
        <setting name="fieldWidth">70</setting>
        <setting name="fieldHeight">6</setting>
        <setting name="minChars">0</setting>
        <setting name="maxChars">65535</setting>
      </settings>
      <typeRestrictions>
        <restriction code="r1">
          <table>ca_objects</table>
          <settings>
            <setting name="minAttributesPerRow">0</setting>
            <setting name="maxAttributesPerRow">255</setting>
            <setting name="minimumAttributeBundlesToDisplay">1</setting>
          </settings>
```

(continues on next page)

(continued from previous page)

```
        </restriction>
        <restriction code="r2">
          <table>ca_places</table>
          <settings>
            <setting name="minAttributesPerRow">0</setting>
            <setting name="maxAttributesPerRow">255</setting>
            <setting name="minimumAttributeBundlesToDisplay">1</setting>
          </settings>
        </restriction>
      </typeRestrictions>
    </metadataElement>
```

### DataTypes (AttributeTypes)

Each element in an element set must be declared with a specific datatype (sometimes referred to as attribute types). These configure what kind of data will be entered into a specific element, how it will be formatted, and how it will be stored. At this time there are 20 Attribute Types to chose from.

Lists are unique because they require reference code names defined in your list definitions. For example:

```
<metadataElement code="dcFormat" datatype="List" list="dc_format">
      <labels>
        <label locale="en_US">
          <name>Format</name>
          <description>The file format or physical medium or dimensions of␣
↪the resource.</description>
        </label>
      </labels>
      <documentationUrl>http://dublincore.org/documents/dcmi-terms/#terms-
↪format</documentationUrl>
```

In this element set the code name is dcFormat, its datatype is List, the preferred labels are in English it is "Format". A description has been added and will appear as help text in the cataloging interface when the label is moused over. The list is defined from a code name defined in the list definitions. In this particular case it's dc_format. This links the element set with its corresponding List in list definitions creating the necessary code for the controlled vocabulary values. See List and Vocabulary Management for more details.

### Settings

Next, the settings define how and what the data field will display. Settings vary based upon the datatype of the element. For a text element settings include width, height, maximum and minimum characters allowed. Refer to Attribute Types for specific settings.

If you would like to make a particular text field a required data entry point, set your minimum characters to 1. This will require at least 1 character to be entered in the field before the element set can be saved.

When defining list elements, you can use the render setting to control whether the list is displayed as a drop-down menu (render=select); radio buttons (render=radio_buttons); Yes/no checkbox (render=yes_no_checkboxes); a checklist (render=checklist); type-ahead lookup (render=lookup); horizontal hierarchy browser *(render=horiz_hierbrowser)*; horizontal hierarchy browser with search *(render=horiz_hierbrowser_with_search)*; or vertical hierarchy browser *(render=vert_hierbrowser)*. More information on List settings can be found here.

To configure the sort order of list items, you can use the defaultSort setting. This setting using numeric codes: Sort by label = 0; Sort by rank field = 1; Sort by value field = 2; and Sort by idno = 3.

### Type Restrictions

Type restrictions do pretty much what they sound like they do - they restrict element sets according to types. More precisely they restrict element sets to specific item types (objects, entities, places, occurrences, etc.) and define how attributes can be created and displayed. It is possible to target a restriction to an item type in general or to a specific type value for an item. For example, if you have defined an object type (in the object_types system list) of “periodical”, you can restrict an element set to be valid for only objects that are periodicals. Attributes of that element set will appear on editing forms only for periodicals (and whatever else it is bound to via type restrictions). Type restrictions can be assigned to as many item types as necessary for that particular element set.

Defining a restriction by, for example, object types creates unique cataloging interfaces based on the kinds of objects that you have in your collection. For instance, you may want to have different data entry fields for multimedia content than you want for paper-based materials. By defining these object types in the list definitions and then linking them to “type” under type restrictions, you can create object type-specific cataloging interfaces.

```
<typeRestrictions>
    <restriction code="r1">
        <table>ca_objects</table>
        <type>moving_images</type>
        <settings>
            <setting name="minAttributesPerRow">0</setting>
            <setting name="maxAttributesPerRow">255</setting>
            <setting name="minimumAttributeBundlesToDisplay">1</setting>
        </settings>
    </restriction>
</typeRestrictions>
```

**Note:** Note that the item type specification - what type of item the restriction is bound to - is called “table” in the profile code. This is because the item types are specified using the names for their tables in the CollectiveAccess database. Specific table names are used for these Primary Types.

Each type restriction can take settings. Currently defined settings values are:

| Setting | Value |
|---|---|
| minAttributesPerRow | Minimum number of attributes of this kind that must be associated with an item; set to zero to make the attribute optional; valid value must be a positive integer. |
| maxAttributesPerRow | Maximum number of attributes of this kind that can be associated with an item; set to zero or do not set to enforce no limit; valid value must be a positive integer. Setting value to 1 will disable the repeat attribute option in the editor screen, set to 0 or greater than one will enable the attribute repeat feature and display the “Add” link (eg. [+]Add <attr_name> ) immediately below the attribute data entry. |
| minimumAttributeBundlesToDisplay | The minimum number of attribute bundles to show in an editing form. If the number of actual attributes is less than this number then the user interface will show empty form bundles to reach this number. This number should be less than or equal to the maximum number of attributes per row; valid values are positive integers. |

### User Interface Definitions

User interface definitions configure the layout of element sets within the cataloging system. Here you can bring together all the element sets and arrange them into a manageable cataloging interface through designation of Screens and Bundles.

Screens are used to group metadata attributes and create a desired cataloging workflow. Bundles are user interface elements that can be placed on each screen. They can be editable attributes of a specific element set or editable database fields intrinsic to a specific item type. Or they can be user interfaces that allow cataloguers to establish relationships with other items, add and remove items from sets and manage an item's location in a larger hierarchy. Bundles are so named because they are essentially black-boxes that encapsulate various functionality. You don't need to know how they implement this functionality. You need only place them where you want them to be.

That user interfaces are just ordered arrangements of form elements and controls - bundles - makes them highly configurable. Perhaps you want only Title and ID on the first screen, Basic info, and additional data on the second screen, Additional info, and multimedia on the third, Media. In Figure 2 you can see the various tabs in the left side navigation in Providence. These tabs are actually defined as screens in the user interface definitions found in the installation profile. Note that the Summary and Log tabs seen below in Figure 2 are system screens that appear automatically. They allow you to display data and change logs associated with the record.

Figure 2 : Left side nav.png

To organize the data fields on each screen, you must first declare which interface you are working within with a unique code and table type. After the editor is defined, you can begin to create and fill up screens.

```
<userInterface code="standard_object_ui" type="ca_objects">
      <labels>
        <label locale="en_US">
          <name>Standard object editor</name>
        </label>
      </labels>
      <screens>
        <screen idno="basic" default="1">
          <labels>
            <label locale="en_US">
              <name>Basic info</name>
            </label>
          </labels>
          <bundlePlacements>
            <placement code="idno">
              <bundle>idno</bundle>
            </placement>
            <placement code="preferred_labels">
              <bundle>preferred_labels</bundle>
              <settings>
                <setting name="label" locale="en_US">Title</setting>
                <setting name="add_label" locale="en_US">Add title</setting>
              </settings>
            </placement>
            <placement code="nonpreferred_labels">
              <bundle>nonpreferred_labels</bundle>
              <settings>
                <setting name="label" locale="en_US">Alternate titles</
↪setting>
                <setting name="add_label" locale="en_US">Add name</setting>
              </settings>
            </placement>
            <placement code="ca_attribute_date">
              <bundle>ca_attribute_date</bundle>
            </placement>
          </bundlePlacements>
        </screen>
```

**Note:** Note that each entry in the bundle list is actually comprised of several parts: a unique code as key and an associative array as value. At a minimum, the value array must define a bundle using the 'bundle' key and a valid bundle name as the value. Depending upon the bundle being listed, other settings can be passed as well.

---

The list of valid bundles varies according to the type of item being edited. The object editor supports certain bundles that the entities editor does not. All editors support attribute bundles. To derive the bundle name for a specific element_set simply preface the element code with *ca_attribute_* For example, the element_set creation_date would have a bundle name of *ca_attribute_creation_date*. The bundle names for various intrinsic database fields are the field names themselves. These Intrinsic Bundles names for other user interface elements are unique to the Primary Types.

Each bundle you add to a user interface can take optional settings depending on the bundle type. A full list of these settings is defined on the Bundles page.

The label and add_label allow you to override the default text labels in the user interface that are output above a bundle and on the add button respectively. Each of these settings is an associative array with locale codes as keys and the label text to use as values. Including both English and German display text translations would look like this:

```
<placement code="ca_attribute_description">
   <bundle>ca_attribute_description</bundle>
        <settings>
          <setting name="label" locale="en_US">Narrative description</
↪setting>
          <setting name="label" locale="de_DE">Beschriebung</setting>
          <setting name="add_label" locale="en_US">Add another description</
↪setting>
          <setting name="add_label" locale="de_DE">Addieren einen␣
↪Beschriebung</setting>
        </settings>
</placement>
```

The restrict_to_type setting applies only to bundles that create relationships between items - bundles like ca_objects and ca_entities. By default these bundles will allow linking to an item of any type - for example, by default ca_entities will let you link to an individual, and organization or any other type of entity. restrict_to_type does just what it says, which is limit the bundle such that it allows linking only to a specific type. The value you set restrict_to_type to should be the identifier (idno field value) of the type you wish to restrict to.

For example, if you have an entity type of individual with its list item idno set to ind, then the bundle specification for a ca_entities linking control that only allows linking to individuals would look like this:

```
<bundle>ca_entities</bundle>
        <settings>
        <setting name="restrict_to_types">ind</setting>
        </settings>
```

Additionally, the restrtict_to_relationship_types setting applies only to bundles that create relationships between items - bundles like ca_objects and ca_entities, but need to be linked to a specific relationship type. By default, when you create a relationship between any two records, that relationship can be defined through Relationship Types (see section 3.2.6). Furthermore, those relationship types can have hierarchical subtypes and so on. This is where being able to restrict to a relationship type is useful.

For example, let's say you have entities relationship types that are creator, contributor, and publisher. These are further defined by another level of sub-types, so you can further define creator as artist, author, director, etc; contributor as assistant director, production assistant, etc; and publisher as copyright holder, distributor, etc. You can use restrict_to_relationship_type to create a relationship field limited only to that type. To create a relationship field that would only be limited to creator types, it would look like this:

```
<placement code="ca_creator">
       <bundle>ca_entities</bundle>
```

(continues on next page)

(continued from previous page)

```
        <settings>
        <setting name="restrict_to_relationship_types">creator</setting>
        <setting name="label" locale="en_US">Creators & contributors</setting>
        <setting name="add_label" locale="en_US">Add creator & contributor</
↪setting>
        </settings>
      </placement>
```

As of version 1.7 you can restrict display of bundle placements on a screen depending upon the type of the record being edited. For example, you can set up a bundle that appears when editing objects of type “book” but not type “video”. By default a bundle will display for all types. To restrict it, set the typeRestrictions attribute in the <placement> tag to a comma-separated list of type codes. For example:

```
<placement code="ca_creator" typeRestrictions="book,document">...</placement>
```

### Relationship Types

CollectiveAccess creates relationships between records that are qualified by descriptive types. These types for Relationships create the necessary language to describe relationships between items in the cataloging interface, from the point of view of either item. For example an entity can be a “creator” of an object, and an object can be “created” by an entity. Each possible relationship in CollectiveAccess has its own list of relationship types. You must define at least one type for each relationship. Relationships with no defined types will not be usable.

When specifying relationship types in a profile, you must specify to which relationship each type belongs. Each has a unique name, which is actually the name of the underlying database table that stores the relationship data. The naming of these tables follows a simple pattern: the names of the two items related connected by “_x_” and prefixed with *ca_*. Thus the name of the object to entity relationship is ca_objects_x_entities.

However, it’s not quite so simple; you can’t just guess the names without resorting to a list. The order of the two item names matters, but does not follow a clearly predictable pattern. ca_objects_x_entities works but ca_entities_x_objects doesn’t. Therefore, the naming Relationships is critical.

Relationships manifest themselves in the cataloging interface as repeating bundles that consist of:

- A relationship type drop-down to qualify the relationship
- An autocompleting lookup into the related authority
- An optional date range qualifier
- Optional attributes (generally text, but could include other types of data)
- Optional reification - relationships between the relationship on other authority items

Note that the date range qualifier, optional attributes and reification are not implemented in the user interface yet. Relationship types bundles are typically expressed like this.

```
<relationshipTable name="ca_objects_x_entities">
      <types>
        <type code="creator" default="1" rank="1">
          <labels>
            <label locale="en_US">
              <typename>created by</typename>
              <typename_reverse>is creator</typename_reverse>
            </label>
          </labels>
          <subTypeLeft> </subTypeLeft>
```

(continues on next page)

(continued from previous page)

```
            <subTypeRight/>
          </type>
          <type code="publisher" default="0" rank="3">
            <labels>
              <label locale="en_US">
                <typename>published by</typename>
                <typename_reverse>is publisher</typename_reverse>
              </label>
            </labels>
            <subTypeLeft> </subTypeLeft>
            <subTypeRight/>
          </type>
          <type code="contributor" default="0" rank="2">
            <labels>
              <label locale="en_US">
                <typename>contributed by</typename>
                <typename_reverse>is contributor</typename_reverse>
              </label>
            </labels>
            <subTypeLeft> </subTypeLeft>
            <subTypeRight/>
          </type>
```

In this example the relationship types are ordered as follows: creator, contributor, publisher. This order is established by the rank setting in the type tag.

### Putting It All Together

To briefly summarize the components of installation profiles:

- Locales define languages for translations and these codes are used throughout the profile under "preferred_labels"
- List Definitions create system lists, user lists and vocabularies. - System Lists determine the types of objects, entities, places, etc., that will display in your system and their sources. System types can also be used to restrict metadata element (attribute) sets to a specific type. - User Lists create simple controlled vocabularies. User lists are used in metadata element (attribute) sets to reference a specific list of types when the DataType "List" is used. - Vocabularies create multi-level controlled vocabularies, the terms of which can be related to other items for descriptive cataloguing.
- Metadata Element (Attribute) Sets configure the look and function of data entry fields in your cataloging system. They must be defined by one of several AttributeTypes. Metadata Element (Attribute) Sets are referenced in Bundles to design Screens in User Interface Definitions.
- User Interface Definitions layout the cataloging navigation in Providence. These are divided into one or more screens, each of which contains one or more bundles. - Bundles are user interface elements that can be placed on each screen. They can be editable attributes of a specific metadata element set or editable database fields intrinsic to a specific item type. Or they can be user interface elements that allow cataloguers to establish relationships with other items, add and remove items from sets and manage an item's location in a larger hierarchy.
- Relationships create the relational structure and language between items. - Relationship types qualify relationships

### Modifying an Existing Profile

The simplest way to create a custom cataloging interface is to modify an existing profile. You will find in the Configuration Library at CollectiveAccess.org profiles implementing many different metadata standards and suitable for a wide range of projects. Select the profile that most closely matches your collection or cataloging project. Review the documentation on that profile and determine what changes, if any, need to be made.

Next, map the required changes to the five profile sections. Do you need a translation? Then set up a locale. A new list of media formats? Set up a new list definition. Create a new element set for that list and then add it to a bundle in user interface definitions.

Modifying an existing profile will help to ensure that all the necessary components for a functioning installation profile are present and working. It also saves a whole lot of typing!

### Building a Profile from Scratch

It is highly recommended that you modify an existing profile, but if for some reason you find that none of the existing profiles meets the needs of your project then you may need to build a new profile from scratch.

### Begin with the Basics

Before coding a new profile, it's very important to think about how you want your catalog to function. Ask yourself these questions:

- What kind of objects are in my collection?
- What information do I need to organize and find these objects?
- How do I want this information be structured?

Try creating a spreadsheet with all the elements (data entry fields) you want to have in your system. Include a definition for each element, how it will be used, if any special functions are needed (controlled vocabularies, dropdown menus, etc), if the element will be required, and any other details you think are necessary to note.

### Study other Profiles and the Wiki

Another extremely helpful exercise to go through before coding a new profile is to closely analyze existing profiles for their layout, patterns, and format. The xml files for profiles available in your CollectiveAccess installation can be viewed at:

```
install/profiles/xml/nameofprofile.xml
```

This wiki is also a great resource. It may be helpful to review more in-depth technical documentation about the topics discussed in this introductory guide.

### Work in Sections

There are 5 sections that open and close with every part of the profile (i.e. <locales>, <elementSets>, etc.) and each uses aspects from another section or builds upon the next section. Begin with the first section, and move through one part at time taking note of code names and formatting as you go along. Working slowly and carefully will pay off later during installation and testing.

#### Saving a Profile

Save your file in:

```
install/profiles/xml
```

Once saved, your new profile should display in the dropdown menu on the installation webpage.

### 1.7.4 Definition and Purpose of a Profile

A CollectiveAccess installation profile is an XML document that tells the software how to set up various aspects of Providence at the time of its installation. The profile enables you to configure nearly every aspect of the various cataloging interfaces in CollectiveAccess before you begin using the system. After installation, you can easily make additional changes using the tools in the "Manage" menu, but it's usually more efficient to set up your installation in such a way that it meets your requirements from the start. Installation profiles:

- Generate and define controlled vocabularies
- Define and label metadata fields
- Specify the method of metadata entry (e.g. a text entry field or a drop-down menu)
- Bundle the fields together for easy metadata entry
- Combine metadata elements on different screens for workflow management
- Delineate and describe the relationships between all of the various types of objects, entities, occurrences, lots, sets, etc. in your system
- Set the logins for different user types
- Configure the display of search results and data exports.
- Create standards-compliant set-ups

Installation profiles "live" in the 'install/profiles/xml' folder located in the directory where you loaded the software on your computer or server. When you first log in after installing CollectiveAccess, you'll be asked to select one of the saved installation profiles in the "XML" folder, which will then be used to complete the installation process. In Providence, many pre-defined profiles are available, ranging from standard schemata (see Metadata Standards) to custom set ups created for and by organizations world wide (see Configuration Library).

### 1.7.5 Creating a Profile

Profiles are written using an xml-based syntax. Typically no profile is created from scratch, but rather users modify existing profiles to meet their needs.

### 1.7.6 Troubleshooting Profiles

Installation profiles are often long and complex text documents. It's easy to make mistakes that cause the installation process to fail or deviate from requirements. You can make errors much less likely by validating your profile against the profile syntax XML schema. The schema is located in *install/profiles/xml/profile.xsd*. Simply copy the schema to the same directory as the profile you are editing and use a validating XML editor such as OxygenXML. The editor will highlight mistakes as you type and point you to the location of the errors.

**Tip:** The CollectiveAccess installer will validate your profile against the schema before proceeding with installation, so if a profile doesn't validate during editing it won't be accepted by the installer. The bottom line: always make sure your profile validates!

### 1.7.7 Changing the installation profile of an existing system

An oft-asked question is "I installed my system using installation profile X. How can I now change it to Y?" The answer is you can't. Installation profiles are simply collections of rules (or templates, if you prefer) for the installer to follow when setting up a new system. Once the installation process is complete the profile ceases to play a role. You can continue to modify the configuration of your system using the web-based configuration tools, creating an installation different from the profile that originally created it. If you really need to change an existing system to conform to a new profile you have two choices: (1) modify the existing system by hand using the web-based configuration tools to match or (2) reinstall from scratch with the desired profile. In the latter case you will lose all existing data, of course.

## 1.8 Primary Tables and Intrinsic Fields

CollectiveAccess is structured around several primary tables, with editors that can be enabled (or disabled) depending on project requirements. Each primary table has intrinsic bundles and its own set of preferred and non-preferred labels bundles. Distinct user interfaces can be configured for each table, and within that, a single table can have multiple user interfaces restricted by Type (see Types).

Editors that are not relevant for your system (you don't catalogue places for example) can be disabled in the configuration file app.conf, by setting the various *_disable directives below to a non-zero value

Here's how it looks in app.conf:

```
# Editor "disable" switches
# -------------------
ca_objects_disable = 0
ca_entities_disable = 0
ca_places_disable = 0
ca_occurrences_disable = 0
ca_collections_disable = 0
ca_object_lots_disable = 0
ca_storage_locations_disable = 0
ca_loans_disable = 0
ca_movements_disable = 1
ca_tours_disable = 1
ca_tour_stops_disable = 1
ca_object_representations_disable = 1
```

### 1.8.1 Objects (ca_objects)

Object records represent items or assets in a collection, typically the physical or born-digital items being managed. Every object record has a “type” that determines which fields are relevant for it. The list of types available in your system can be customized to match your specific cataloging requirements.

### Object intrinsics (ca_objects)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The object identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_objects is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_objects is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the object_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | | |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent object may be used. | No | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for “public” and “private” visibility. For historical reasons the value stored in the intrinsic is the list item’s *value* field, not its identifer or label. By convention “0” is interpreted as private and “1” as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item’s *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Lot | lot_id | A reference to the lot record (ca_object_lots) of which the object is a part. May be null if the object is not part of a lot. Note that an object may be part of only one lot. The raw database value contained lot_id is an internally generated numeric lot_id. However, when setting this intrinsic via an import mapping or API call you may also use the lot’s identifier. | No | |
| Source | source_id | A value from the object_sources list indicating the original source of the object. This value is sometimes used to broadly distinguish different classes of objects. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Deaccessioned? | is_deaccessioned | A flag indicating whether the object is deaccessioned. Will be 1 when deaccessioned or 0 (the default) when not deaccessioned. | Yes | 0 |
| Date of deaccession | deaccession_date | The date of deaccession. If unknown the value will be null. The date is stored as an historic daterange and may be any valid historic date (Eg. it is not limited to post-1970 dates). | No | |
| Date of disposal | deaccession_disposal_date | The date of disposal of the object. This is typ- | No | |

*Note:* ca_objects.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_object_labels table

### 1.8.2 Object Lots (ca_object_lots)

Lots record the accession or acquisition of one or more objects. Lots are commonly used by collecting institutions who may accession more than one unique item per accession. Registrarial information, such as the Deed of Gift, may be recorded in a lot record while cataloging for each accessioned object remains at the object level.

### Object lot intrinsics (ca_object_lots)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno_stub | The lot identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_object_lots is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_object_lots is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the object_lot_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent lot may be used. | No | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Source | source_id | A value from the object_lot_sources list indicating the original source of the lot. This value is sometimes used to broadly distinguish different classes of lots. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Accession status | lot_status_id | A value from the object_lot_statuses list indicating the accession status of the lot. Accession status values might include "accessioned", "pending accession", "non-accessioned item", etc. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Extent | extent | The numeric extent. Must a be a whole, positive number. Default is 0. | Yes | 0 |
| Units of extent | extent_units | Units of extent value, as text. | Yes | |
| Submitted by user | submission_user_id | For records submitted via the Pawtucket "contribute" form interface. The user who submitted the record. | No | |
| Submission group | submission_group_id | For records submitted via the Pawtucket "contribute" form interface. The group of the user | No | |

*Note:* ca_object_lots.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_object_lot_labels table

### 1.8.3 Entities (ca_entities)

Entity records represent specific people and organizations. Relationships can be created between entity and object records (or any other records in any other table) with fully customizable relationship types. For example, an entity record for an individual could be related to an object record as the creator of the object, or the photographer, donor, publisher, performer, etc.

### Entity intrinsics (ca_entities)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The entity identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_entities is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_entities is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the entity_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent entity may be used. | No | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Lifespan | lifespan | The life dates of the entity expressed as an historic daterange. | No | |
| Source | source_id | A value from the entity_sources list indicating the original source of the entity. This value is sometimes used to broadly distinguish different classes of entities. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Submitted by user | submission_user_id | For records submitted via the Pawtucket "contribute" form interface. The user who submitted the record. | No | |
| Submission group | submission_group_id | For records submitted via the Pawtucket "contribute" form interface. The group of the user that submitted the record. | No | |
| Submission status | submission_status_id | For records submitted via the Pawtucket "contribute" form interface. A value from the submission_statuses list indicating the review status of the submitted record. | No | |
| Submission form | submission_via_form | For records submitted via the Pawtucket "contribute" form interface. The identifying code of the form used to submit the record. | No | |

*Note:* ca_entities.preferred_labels.displayname is used by data mappings and display templates to reference the intrinsic displayname field in the ca_entity_labels table. See below *ca_entity_labels name fields* for all ca_entity_labels name fields.

### 1.8.4 Places (ca_places)

Place records represent physical locations, geographic or otherwise. Places are inherently hierarchical allowing you to nest more specific place records within broader ones. As with entities, places can be related records in other tables. Places are typically used to model location authorities specific to your system. For cataloguing of common geographical place names consider using CollectiveAccess' built-in support for GoogleMaps, OpenStreetMap, GeoNames and/or the Getty Thesaurus of Geographic Names (TGN).

### Place intrinsics (ca_places)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The place identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_places is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_places is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the place_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent place may be used. | No | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Lifespan | lifespan | The life dates of the place expressed as an historic daterange. | No | |
| Source | source_id | A value from the places_sources list indicating the original source of the place. This value is sometimes used to broadly distinguish different classes of places. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Floorplan | floorplan | Uploaded image depicting floor plan of place. Used as the base layer in the object-place floorplan user interface. | No | |
| Submitted by user | submission_user_id | For records submitted via the Pawtucket "contribute" form interface. The user who submitted the record. | No | |
| Submission group | submission_group_id | For records submitted via the Pawtucket "contribute" form interface. The group of the user that submitted the record. | No | |
| Submission status | submission_status_id | For records submitted via the Pawtucket "contribute" form interface. A value from the submission_statuses list indicating the review status of the submitted record. | No | |

*Note:* ca_places.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_place_labels table

### 1.8.5 Occurrences (ca_occurrences)

Occurrences are used to represent temporal concepts such as events, exhibition, productions or citations.

### Occurrence intrinsics (ca_occurrences)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The occurrence identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_occurrences is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_occurrences is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the occurrence_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent occurrence may be used. | No | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Source | source_id | A value from the occurrence_sources list indicating the original source of the occurrence. This value is sometimes used to broadly distinguish different classes of occurrences. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Submitted by user | submission_user_id | For records submitted via the Pawtucket "contribute" form interface. The user who submitted the record. | No | |
| Submission group | submission_group_id | For records submitted via the Pawtucket "contribute" form interface. The group of the user that submitted the record. | No | |
| Submission status | submission_status_id | For records submitted via the Pawtucket "contribute" form interface. A value from the submission_statuses list indicating the review status of the submitted record. | No | |
| Submission form | submission_via_form | For records submitted via the Pawtucket "contribute" form interface. The identifying code of the form used to submit the record. | No | |
| View count | view_count | Number of times record has been viewed in Pawtucket front-end | No | 0 |

*Note:* ca_occurrences.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_occurrence_labels table

### 1.8.6 Collections (ca_collections)

Collections represent significant groupings of objects. They may refer to physical collections, symbolic collections of items associated by some criteria, or any other grouping. Collection records are often used to manage formal archival processing and the creation of finding aids, by configuring records to be compliant with the Describing Archives (DACS) content standard.

*Note:* ca_collections.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_collection_labels table

### Collection intrinsics (ca_collections)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The collection identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_collections is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_collections is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the collection_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent collection may be used. | No | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for “public” and “private” visibility. For historical reasons the value stored in the intrinsic is the list item’s *value* field, not its identifer or label. By convention “0” is interpreted as private and “1” as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item’s *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Source | source_id | A value from the collection_sources list indicating the original source of the collection. This value is sometimes used to broadly distinguish different classes of collections. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Submitted by user | submission_user_id | For records submitted via the Pawtucket “contribute” form interface. The user who submitted the record. | No | |
| Submission group | submission_group_id | For records submitted via the Pawtucket “contribute” form interface. The group of the user that submitted the record. | No | |
| Submission status | submission_status_id | For records submitted via the Pawtucket “contribute” form interface. A value from the submission_statuses list indicating the review status of the submitted record. | No | |
| Submission form | submission_via_form | For records submitted via the Pawtucket “contribute” form interface. The identifying code of the form used to submit the record. | No | |
| View count | view_count | Number of times record has been viewed in Pawtucket front-end | No | 0 |

### 1.8.7 Storage Locations (ca_storage_locations)

Storage location records represent physical locations where objects may be located, displayed or stored. Like place records, storage locations are hierarchical and may be nested to allow notation location at various levels of specificity (building, room, cabinet, drawer, etc.). As with the other primary tables, each storage location may have arbitrarily rich cataloguing, including access restrictions, geographical coordinates, keywords and other information.

### Storage location intrinsics (ca_storage_locations)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The storage location identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_storage_locations is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_storage_locations is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the storage_location_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent storage location may be used. | No | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Source | source_id | A value from the storage_location_sources list indicating the original source of the storage location. This value is sometimes used to broadly distinguish different classes of storage locations. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Icon | icon | Icon image to display for storage location. | No | |
| Color | color | Highlight color for storage location in hex format. | No | |
| Is enabled? | is_enabled | Flag indicating whether storage location is available for use (value set to 1) or not available (value is 0). | Yes | 0 |
| Submitted by user | submission_user_id | For records submitted via the Pawtucket "contribute" form interface. The user who submitted the record. | No | |
| Submission group | submission_group_id | For records submitted via the Pawtucket "contribute" form interface. The group of the user that submitted the record. | No | |
| Submission status | submission_status_id | For records submitted via the Pawtucket "contribute" form interface. A value from the sub- | No | |

*Note:* ca_storage_locations.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_storage_location_labels table

### 1.8.8 Loans (ca_loans)

Loan records record details of both incoming and outgoing loans of objects. Loan records, like those in all other tables, is fully customizable and can be used to track alls aspects of a loan, including dates, shipping, and insurance information.

### Loan intrinsics (ca_loans)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The loan identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_loans is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_loans is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the loan_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent loan may be used. | No | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Source | source_id | A value from the loan_sources list indicating the original source of the entity. This value is sometimes used to broadly distinguish different classes of entities. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Submitted by user | submission_user_id | For records submitted via the Pawtucket "contribute" form interface. The user who submitted the record. | No | |
| Submission group | submission_group_id | For records submitted via the Pawtucket "contribute" form interface. The group of the user that submitted the record. | No | |
| Submission status | submission_status_id | For records submitted via the Pawtucket "contribute" form interface. A value from the submission_statuses list indicating the review status of the submitted record. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Submission form | submission_via_form | For records submitted via the Pawtucket "contribute" form interface. The identifying code of the form used to submit the record. | No | |

*Note:* ca_loans.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_loan_labels table

### 1.8.9 Movements (ca_movements)

For more complex location tracking needs, movement records can be used to record in precise detail movement of objects between storage locations, while on loan or while on exhibition. Used as part of a location tracking or use history policy, movements can provide a robust record of every movement event in an object's history.

### Movements intrinsics (ca_movements)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The movement identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_movements is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_movements is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the movement_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Source | source_id | A value from the movement_sources list indicating the original source of the movement. This value is sometimes used to broadly distinguish different classes of mivements. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Submitted by user | submission_user_id | For records submitted via the Pawtucket "contribute" form interface. The user who submitted the record. | No | |
| Submission group | submission_group_id | For records submitted via the Pawtucket "contribute" form interface. The group of the user that submitted the record. | No | |
| Submission status | submission_status_id | For records submitted via the Pawtucket "contribute" form interface. A value from the submission_statuses list indicating the review status of the submitted record. | No | |
| Submission form | submission_via_form | For records submitted via the Pawtucket "contribute" form interface. The identifying code of the form used to submit the record. | No | |
| View count | view_count | Number of times record has been viewed in Pawtucket front-end | No | 0 |

*Note:* ca_movements.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_movement_labels table

### 1.8.10 Object Representations (ca_object_representations)

Representations capture representative digital media (images, video, audio, PDFs) for objects. Representation records usually contain only just a media file, but can accommodate additional cataloguing that is specific to the media file (not to the object the file depicts or represents) if desired. When used. representation metadata often includes captions, credits, access information, rights and reproduction restrictions.

### Object representation intrinsics (ca_objects_representations)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The representation identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_object_representations is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_object_representations is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the object_representation_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| MD5 checksum | md5 | The MD5 checksum of the original media uploaded to the represenatation. | Yes | |
| MIME type | mimetype | The MIME type of the original media uploaded to the representation. Ex. for a JPEG image the MiME type will be image/jpeg. For a PDF the MIME type will be application.pdf. | Yes | |
| Original filename | original_filename | The file name of the original media uploaded to the representation. For web browser uploads this file name is sent by the client and may not always be defined. | Yes | |
| Media | media | The original uploaded media and derivatives. | Yes | |
| Media metadata | media_metadata | EXIF, IPTC and XMP extracted from the original uploaded media | Yes | |
| Media content | media_content | Text content extracted from the original uploaded media. For PDF and Microsoft Office documents this will be the full text of the document. It will be blank for most other file formats. | Yes | |
| Source | source_id | A value from the entity_sources list indicating the original source of the entity. This value is sometimes used to broadly distinguish different classes of object representations. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Submitted by user | submission_user_id | For records submitted via the Pawtucket "con- | No | |

*Note:* ca_objects_representations.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_objects_representation_labels table

## 1.8.11 Tours (ca_tours)

Tour records capture information about on-site or online tours of objects, locations, collections or any other record in the database.

### Tour intrinsics (ca_tours)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Tour code | tour_code | The tour identifier. Must be a unique alphanumeric code without spaces or punctuation beyond underscores. | Yes | |
| Type | type_id | A value from the tour_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Source | source_id | A value from the tour_sources list indicating the original source of the tour. This value is sometimes used to broadly distinguish different classes of tours. When setting this value in a data import or via an API call the item identifier may be used. | No | |
| Icon | icon | Icon image to display for tour. | No | |
| Color | color | Highlight color for tour in hex format. | No | |
| View count | view_count | Number of times record has been viewed in Pawtucket front-end | No | 0 |
| Rank | rank | The sort order position of the tour. Must be a whole number; lower numbers indicate higher ranking in sort. | Yes | 0 |

*Note:* ca_tours.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_tour_labels table

### 1.8.12 Tour Stops (ca_tour_stops)

Each tour record has any number of ordered "stops". Each tour stop contains metadata about the stop (descriptive text, geographic coordinates, etc.) as well as relationships to relevant objects, entities and more.

### Tour stop intrinsics (ca_tour_stops)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The tour stop identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_tour_stops is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_tour_stops is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the tour_stop_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent place may be used. | No | null |
| Tour | tour_id | A reference to the tour record (ca_tours) of which the stop is a part. Note that a stop is always part of a tour. It cannot exist outside of a tour. The raw database value contained tour_id is an internally generated numeric tour_id. However, when setting this intrinsic via an import mapping or API call you may also use the list's code. | No | |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Icon | icon | Icon image to display for tour stop. | No | |
| Color | color | Highlight color for tour stop in hex format. | No | |
| Rank | rank | The sort order position of the tour stop. Must be a whole number; lower numbers indicate higher ranking in sort. | Yes | 0 |

*Note:* ca_tour_stops.preferred_labels.name is used by data mappings and display templates to reference the intrinsic name field in the ca_tour_stop_labels table

### 1.8.13 Label Tables

Labels are record names or titles. All primary tables have companion label tables. Labels come in two varieties: preferred and non-preferred. Each record has one, and only one, preferred label. The preferred label is used as the record's default display title. Records may have any number of non-preferred labels, which are taken as alternative titles and may be used in searches. Labels are always present and do not need to be configured to exist.

The following shorthand is commonly used to reference preferred labels: <tablename>.preferred_labels.<label table name field>. For example the following would display an object preferred label:

```
ca_objects.preferred_labels.name
```

See label name fields below for table specific name fields.

### 1.8.14 Label Table Intrinsics

Occassionally label table names and intrinsic fields need to be referenced directly, for example while configuring searching indexing. Search indexing in Search_indexing.conf.

**Note:** <table name>.preferred_labels.<name of intrinsic> is used by data mappings and display templates to reference the intrinsic _name_ field for preferred labels. The _<table name>.preferred_labels_ construct is simply an alias for the label table, filtered to return only those entries with the _is_preferred_ set. For example _ca_objects.preferred_labels.name_ and _ca_object_labels.name_ refer to the same thing, except that the _ca_object_labels.name_ version will return _all_ labels, while _ca_objects.preferred_labels.name_ will return only those marked as preferred. Similarly, _<table name>.nonpreferred_labels.<name of intrinsic>_ will return all entries _not_ marked as preferred. Whether you use _ca_objects.preferred_labels.<name of intrinsic>_, ca_objects.nonpreferred_labels.<name of intrinsic>_ or _ca_object_labels.<name of intrinsic>_, the intrinsic names used are the same ones listed below.

#### Label tables for primary table

| Primary table | Label table |
|---|---|
| ca_objects | ca_object_labels |
| ca_object_lots | ca_object_lot_labels |
| ca_entities | ca_entity_labels |
| ca_places | ca_place_labels |
| ca_occurrences | ca_occurrence_labels |
| ca_collections | ca_collection_labels |
| ca_storage_locations | ca_storage_location_labels |
| ca_loans | ca_loan_labels |
| ca_movements | ca_movement_labels |
| ca_object_representations | ca_object_representation_labels |
| ca_tours | ca_tour_labels |
| ca_tour_stops | ca_tour_stop_labels |

### Available for all label tables

| Name | Code | Description |
|---|---|---|
| Preferred? | is_preferred | A preferred label is the one 'true' title or name of an item – the one you should use when referring to the item – used for display. There can only be one preferred label per item per locale. That is, if you are cataloguing in three languages you can have up to three preferred labels, one in each language. Non-preferred labels are alternative names that can be used to enhance searching or preserve identity. Non-preferred labels can repeat without limit, take locales and optionally take type values which may be employed distinguish valid 'alternate' labels from simple search enhancing non-preferred labels. |
| Name sort | name_sort | Automatically generated version of label used for sorting. |
| Type | type_id | |
| Source | source_info | |
| Locale | locale_id | Locale of the label. |

Note: ca_tour_labels and ca_tour_stop_labels do not contain type, source_info and is_preferred

### Label name fields

Name fields within label tables can differ for different tables.

The following applies to: Object labels (ca_object_labels), Object Lot labels (ca_object_lot_labels), Place labels (ca_place_labels), Occurrence labels (ca_occurrence_labels), Collection labels (ca_collection_labels), Storage location labels (ca_storage_location_labels), Loan labels (ca_loan_labels), Movement labels (ca_movement_labels), Object representation labels (ca_object_representation_labels), Tour labels (ca_tour_labels), Tour stop labels (ca_tour_stop_labels)

| Name | Code | Description |
|---|---|---|
| Name | name | Name of record, used for display. |

### The following applies to: Entity labels (ca_entity_labels)

| Name | Code | Description |
|---|---|---|
| Displayname | displayname | Full name of entity, used for display. |
| Forename/First name | forename | Forename of the entity |
| Additional forenames/ first names | other_forename | Alternate forenames |
| Middle name | middlename | Middle name of the entity |
| Surname/Last name | surname | Surname of the entity |
| Prefix | prefix | Prefix for the entity |
| Suffix | suffix | Suffix for the entity |

### 1.8.15 Special Intrinsics

Additional intrinsics provide access to change log information, origination and history tracking information. They are potentially available many or all primary tables, as noted below.

| Code | Description | Applies to | Examples |
|---|---|---|---|
| created | Date/time record was created. Returns date/time formatted using system defaults for display. Optional subfields may be specified to obtain the date/time in different formats, and information about the user that created the record. Subfields include:<br>user = the creator's user name fname = the creator's first name lname = the creator's last name email = the creator's email address timestamp = the creation date/time as a Unix timestamp | Any record | ca_objects.created (returns date/time as display text) ca_objects.created.timestamp (returns date/time as Unix timestamp) ca_objects.created.email (returns the email of the user who created the record) |
| lastModified | Date/time record was last modified. Returns date/time formatted using system defaults for display. Optional subfields may be specified to obtain the date/time in different formats, and information about the user that last modified the record. Subfields include:<br>user = the user name of the user who last modified the record fname = the last modifier's first name lname = the last modifier's last name email = the last modifier's email address timestamp = the last modification date/time as a Unix timestamp | Any record | ca_objects.lastModified (returns date/time of last modification as display text) ca_objects.lastModified.timestamp (returns date/time of last modification as Unix timestamp) ca_objects.lastModified.email (returns the email of the user who last modified the record) |
| _guid | A globally unique identifier (GUID) for the record. This is the same GUID value used to track records across replicated systems. (Available from version 1.7.9) | Any ford | ca_objects.guid |
| history_tracking_current_value | Current value for history tracking policy. Policy used is the default policy unless overridden by passing the a policy option value on the tag. | Any record for which a current value tracking policy is defined | ca_objects.history_tracking_current_value (current value for object default policy) ca_objects.history_tracking_current_value%p (current value for object record using "provenance" policy) |
| history_tracking_current_date | Date of current value for history tracking policy. Policy used is the default policy unless overridden by passing the a policy option value on the tag. | Any record for which a current value tracking policy is defined | ca_objects.history_tracking_current_date (current value cate for object default policy) ca_objects.history_tracking_current_date%po (current value date for object record using "provenance" policy) |
| history_tracking_current_contents | Values of all records that use this record as their current value. Policy used is the default policy unless overridden by passing the a policy option value on the tag. | Any record which is used by at least one current value tracking policy | ca_storage_locations.history_tracking_current (all values that use this location record as their current value for any default policy) ca_storage_locations.history_tracking_current (all records using this location record as their curent value using the "current_location" policy) |
| submitted_by_user | Name and email of user who sub- | ca_objects, | ca_objects.submitted_by_user |

## 1.9 Metadata Elements

### 1.9.1 Attribute Types Settings

#### Attribute settings: Containers

Unlike all other attribute types, containers do not represent data values. Rather their sole function is to organize attributes into groups for display. In a multi-attribute value set (for example an address with separate attributes for street number, city, state, country and postal code), there will be at least one container serving as the "root" (or top) of the attribute hierarchy. Other containers may serve to further group items in the multi-attribute set into sub-groups displayed on separate lines of a form.

| Settings | Description | Default | Values |
|---|---|---|---|
| doesNotTakeLocale | Defines whether element take locale specification. | 0 (takes locale) | 0 or 1 |
| lineBreakAfterNumberOfElements | Number of metadata elements after which a line break should be inserted. | 0 (no line breaks) | Integers zero or greater |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: ^*my_element_code*. | | Display_Templates |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |
| readonlyTemplate | Layout used when a container value is in read-only mode. If this template is set, existing values are always displayed in read-only mode until you click to edit. This can be used to preserve screen space for large containers.<br>Element code tags prefixed with the ^ character, used to represent the value in the template. For example: ^ca_objects.my_notes. Each of these templates is evaluated relative to a specific value instance for this container, so other display template elements like <unit>, <ifdef> or <more> might not work exactly as expected. More general notes on display templates are here: Display_Templates | | HTML |

### Attribute settings: Text

| Settings | Description | Default | Values |
|---|---|---|---|
| minChars | The minimum number of characters to allow. Input shorter than required will be rejected. | 0 (no minimum) | Integers zero or greater |
| maxChars | The maximum number of characters to allow. Input longer than required will be rejected. | 65535 | Integers greater than zero |
| regex | A Perl-format regular expression with which to validate the input. Input not matching the expression will be rejected. Do not include the leading and trailling delimiter characters (typically "/") in your expression. Leave blank if you don't want to use regular expression-based validation. | | Expression-based validation value |
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater than zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater than zero |
| usewysiwygeditor | Check this option if you want to use a word-processor like editor with this text field. If you expect users to enter rich text (italic, bold, underline) then you might want to enable this. | 0 (not enabled) | 0 or 1 |
| default_text | Text to pre-populate a newly created attribute with | | Text |
| sugestExistingValues | Use this option if you want the attribute to suggest previously saved values as text. This option is only effective if the display height of the text entry is equal to 1. | 0 (does not suggest text) | 0 or 1 |
| sugestExistingValueSort | If suggestion of existing values is enabled this option determines how returned values are sorted. Choose valueto sort alphabetically. Choose most recently added to sort with most recently entered values first. | value | value ormost recently added |
| doesNotTakeLocale | Defines whether element take locale specification | 0 (takes locale) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |

### Attribute settings: DateRange

| Settings | Description | Default | Values |
|---|---|---|---|
| dateRangeBoundaries | The range of dates that are accepted. Input dates outside the range will be rejected. Leave blank if you do not require restrictions. | | Accepted date/time expression |
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater than zero |
| useDatePicker | Use this option if you want a calendar-based date picker to be available for date entry. | 0 (not enabled) | 0 or 1 |
| default_text | Value to pre-populate a newly created attribute with. | | Accepted date/time expression |
| mustNotBeBlank | Use this option if this attribute value must be set to some value, if, in other words, it must not be blank. | 0 (can be blank) | 0 or 1 |
| isLifespan | Use this option if this attribute value represents a persons lifespan. Lifespans are displayed in a slightly different format in many languages than standard dates. | 0 (not enabled) | 0 or 1 |
| suggestExistingValues | Use this option if you want the attribute to suggest previously saved values as text. This option is only effective if the display height of the text entry is equal to 1. | 0 (does not suggest text) | 0 or 1 |
| suggestExistingValueSort | If suggestion of existing values is enabled this option determines how returned values are sorted. Choose value to sort alphabetically. Choose most recently added to sort with most recently entered values first. | value | value or most recently added |
| doesNotTakeLocale | Defines whether element take locale specification | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values | , (comma) | Text |

### Attribute settings: Lists

| Settings | Description | Default | Values |
|---|---|---|---|
| listWidth | Width, in characters or pixels, of the list when displayed in a user interface. When list is rendered as a hierarchy browser width must be in pixels. | 40 | Characters or pixels |
| listHeight | Height, in pixels, of the list when displayed in a user interface as a hierarchy browser. | 200px | Pixels |
| maxColumns | Maximum number of columns to use when laying out radio buttons or checklist. | 3 | Integers greater than zero |
| render | Determines how the list is displayed visually. | Drop-down list (‘select’) | Drop-down menu (code=”select”) Yes/no checkbox (code=”yes_no_checkboxes”) Radio buttons (code=”radio_buttons”); Checklist (code=”checklist”) Type-ahead lookup (code=”lookup”) Horizontal hierarchy browser (code=”horiz_hierbrowser”) Horizontal hierarchy browser with search (code=”horiz_hierbrowser_with_search”) Vertical hierarchy browser (code=”vert_hierbrowser”) |
| doesNotTakeLocale | Defines whether element takes locale specification | 1 (does not take locale specifications) | 0 or 1 |
| requireValue | Defines whether a list item must be explicitly set. If set to 1 then a valid list item must be selected, and in the absence of a selected value the default value is used. If set to zero then a “null” value (labeled “none” in the English locale) is added to the list and made default | 1 (require value) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: ^*my_element_code*. | | HTML |
| dis- | Delimiter to use between multiple val- | , | Text |

### Attribute settings: Geocode

The Geocode attribute type represents one or more coordinates (latitude/longitude pairs) indicating the location of an item (collection object, geographic place, storage location or whatever else you want to place on a map).

Coordinates can be entered as decimal latitude/longitude pairs (ex. 40.321,-74.55) or in degrees-minutes-seconds format (ex. 40 23' 10N, 74 30' 5W). Multiple latitude/longitude coordinates should be separated with semicolons (";"). UTM format coordinates is also supported.

Non-coordinate entries are converted to coordinates using the Google Maps Geocoding service, which works well for most full and partial addresses worldwide. To unambiguously distinguish coordinate data from address data to be geocoded, it is strongly suggested that coordinate lists be enclosed in square brackets (ex. [40.321,-74.55; 41.321,-74.55;41.321,-75.55;40.321,-75.55;40.321,-74.55].

### Google Maps integration

The Google Maps mapping service is used to produce maps displaying your coordinates. The Geocode attribute used to support either version 2 or 3 of the Google Maps API, selectable via the google_api directive in the global.conf file. As of August 2010, only the v3 API is supported since v2 is now officially deprecated by Google. Any old v2 configuration in your installation (assuming you installed prior to August 2010) will be ignored.

### Adding rectified overlays

You can add layers that include rectified maps by creating serve-able tiles using a tool such as MapWarper and the Google/OSM URL it provides as an export option. The OSM URL will be in the format `http://mapwarper.net/maps/tile/0001/z/x/y.png`. You will need to change the x, y and z placeholders in ${x}, ${y} and ${z} respectively. The example OSM URL for CollectiveAccess would be `http://mapwarper.net/maps/tile/3671/${z}/${x}/${y}.png`. This URL should be entered into the "Tile Server URL" option for the metadata element. You should also provide a layer name describing the content of the map. If you wish to allow users to toggle the layer on and off check the "Show layer switcher controls" checkbox.

| Settings | Description | Default | Values |
|---|---|---|---|
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 70 | Integers greater than zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 2 | Integers greater than zero |
| mustNotBeBlank | Use this option if this attribute value must be set to some value, if, in other words, it must not be blank. | 0 (can be blank) | 0 or 1 |
| doesNotTakeLocale | Defines whether element takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 0 (not used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| tileServerURL | OSM URL for tileserver to load custom tiles from, with placeholders for X, Y and Z values in the format ${x}. . | http://mapwarper.net/maps/tile/3671/${z}/${x}/${y}.png | URL with placeholders |
| tileLayerName | Display name for layer containing tiles loaded from tile server specified in the tile server URL setting. | 1929 Street Atlas | Text |
| layerSwitcherControl | Include layer switching controls in the map interface. | 1 (show controls) | 0 or 1 |

### Attribute settings: Url

Accepts a properly formatted URL value.

| Settings | Description | Default | Values |
|---|---|---|---|
| minChars | The minimum number of characters to allow. Input shorter than required will be rejected. | 0 (no minimum) | Integers zero or greater |
| maxChars | The maximum number of characters to allow. Input longer than required will be rejected. | 65535 | Integers greater than zero |
| regex | A Perl-format regular expression with which to validate the input. Input not matching the expression will be rejected. Do not include the leading and trailling delimiter characters (typically "/") in your expression. Leave blank if you don't want to use regular expression-based validation. | | Expression-based validation value |
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater than zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater than zero |
| requireValue | Use this option if you want an error to be thrown if the URL is left blank. | 0 (entry not required) | 0 or 1 |
| suggestExistingValues | Use this option if you want the attribute to suggest previously saved values as text. This option is only effective if the display height of the text entry is equal to 1. | 0 (does not suggest text) | 0 or 1 |
| suggestExistingValueSort | If suggestion of existing values is enabled this option determines how returned values are sorted. Choosevalue to sort alphabetically. Choose most recently added to sort with most recently entered values first. | value | value ormost recently added |
| doesNotTakeLocale | Defines whether element takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |
| | | | |

### Attribute settings: Currency

Accepts a currency value composed of a currency specifier and a decimal number.

| Settings | Description | Default | Values |
|---|---|---|---|
| minValue | The minimum value allowed. Input less than the required value will be rejected. | 0 (no minimum) | Numeric |
| maxValue | The maximum value allowed. Input greater than the required value will be rejected. | 0 (no maximum) | Numeric |
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater then zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater then zero |
| doesNotTakeLocale | Defines whether element takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: Length

Accepts length measurements in metric, English and typographical points units.

| Settings | Description | Default | Values |
|---|---|---|---|
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater then zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater then zero |
| requireValue | Use this option if you want an error to be thrown if this measurement is left blank. | 0 (can be blank) | 0 or 1 |
| doesNotTakeLocale | Defines whether measurement takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. |  | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: Weight

Accepts weight measurements in metric and English units.

| Settings | Description | Default | Values |
|---|---|---|---|
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater then zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater then zero |
| requireValue | Use this option if you want an error to be thrown if this measurement is left blank. | 0 (can be blank) | 0 or 1 |
| doesNotTakeLocale | Defines whether measurement takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: TimeCode

Accepts time offsets in a number of time code formats.

| Settings | Description | Default | Values |
|---|---|---|---|
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater than zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater than zero |
| requireValue | Use this option if you want an error to be thrown if this measurement is left blank. | 0 (can be blank) | 0 or 1 |
| doesNotTakeLocale | Defines whether measurement takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: Integer

Accepts a properly formatted integer value.

| Settings | Description | Default | Values |
|---|---|---|---|
| minChars | The minimum number of characters to allow. Input shorter than required will be rejected. | 0 (no minimum) | Integers zero or greater |
| maxChars | The maximum number of characters to allow. Input longer than required will be rejected. | 65535 | Integers greater than zero |
| minValue | The minimum numeric value to allow. Values smaller than required will be rejected. | | Numeric |
| maxValue | The maximum numeric value to allow. Values greater than required will be rejected. | | Numeric |
| regex | A Perl-format regular expression with which to validate the input. Input not matching the expression will be rejected. Do not include the leading and trailing delimiter characters (typically "/") in your expression. Leave blank if you don't want to use regular expression-based validation. | | Expression-based validation value |
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater than zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater than zero |
| mustNotBeBlank | Use this option if you want an error to be thrown if the URL is left blank. | 0 (entry not required) | 0 or 1 |
| doesNotTakeLocale | Defines whether element takes locale specification. | 0 (takes locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: Numeric

Accepts numeric values and strings consisting of optional sign, any number of digits, optional decimal part and optional exponential part.

| Settings | Description | Default | Values |
|---|---|---|---|
| minChars | The minimum number of characters to allow. Input shorter than required will be rejected. | 0 (no minimum) | Integers zero or greater |
| maxChars | The maximum number of characters to allow. Input longer than required will be rejected. | 10 | Integers greater than zero |
| minValue | The minimum numeric value to allow. Values smaller than required will be rejected. | | Integers |
| maxValue | The maximum numeric value to allow. Values greater than required will be rejected. | | Integers |
| regex | A Perl-format regular expression with which to validate the input. Input not matching the expression will be rejected. Do not include the leading and trailing delimiter characters (typically "/") in your expression. Leave blank if you don't want to use regular expression-based validation. | | Expression-based validation value |
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater than zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater than zero |
| mustNotBeBlank | Use this option if you want an error to be thrown if the URL is left blank. | 0 (entry not required) | 0 or 1 |
| doesNotTakeLocale | Defines whether element takes locale specification. | 0 (takes locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: LCSH

Library of Congress Subject Heading values.

| Settings | Description | Default | Values |
|---|---|---|---|
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 60 | Integers greater than zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater than zero |
| doesNotTakeLocale | Defines whether element takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 0 (not used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to | | HTML |

### Attribute settings: GeoNames

Represents one or more latitude/longitude coordinates

| Settings | Description | Default | Values |
|---|---|---|---|
| field-Width | Width, in characters, of the field when displayed in a user interface. | 60 | Integers greater than zero |
| field-Height | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater than zero |
| mustNot-BeBlank | Use this option if this attribute value must be set to some value, if, in other words, it must not be blank. | 0 (can be blank) | 0 or 1 |
| doesNot-TakeLo-cale | Defines whether element takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| can-BeEmpty | Use this option if you want to allow empty attribute values. This – of course – only makes sense if you bundle several elements in a container. | 0 | 0 or 1 |
| can-BeUsedIn-Sort | Use this option if this attribute value can be used for sorting of search results. | 0 (not used in sort) | 0 or 1 |
| can-BeUsedIn-Search-Form | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| can-BeUsedInDis-play | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| dis-playTem-plate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| display-Delimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: Files

Uploaded file

| Settings | Description | Default | Values |
|---|---|---|---|
| doesNotTakeLocale | Defines whether element takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: Media

Uploaded media (image, sound video).

| Settings | Description | Default | Values |
|---|---|---|---|
| doesNotTakeLocale | Defines whether element takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: Taxonomy

| Settings | Description | Default | Values |
|---|---|---|---|
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 60 | Integers greater then zero |
| doesNot-TakeLocale | Defines whether measurement takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| restrict-ToOccur-renceType-Idno | Insert idno of a occurrence type here to restrict the lookup mechanism to that type. | | idno |
| can-BeUsedIn-Sort | Use this option if this attribute value can be used for sorting of search results. | 0 (not used in sort) | 0 or 1 |
| can-BeUsedIn-Search-Form | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| can-BeUsedInDis-play | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| dis-playTem-plate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDe-limiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: Entities

| Settings | Description | Default | Values |
|---|---|---|---|
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 60 | Integers greater then zero |
| doesNotTakeLocale | Defines whether measurement takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |

### Attribute settings: Color

Stores color values. User interface typically provides a color picker. Values are stored internally as RGB hex color values.

| Settings | Description | Default | Values |
|---|---|---|---|
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater then zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater then zero |
| requireValue | Use this option if you want an error to be thrown if this measurement is left blank. | 0 (can be blank) | 0 or 1 |
| doesNotTakeLocale | Defines whether measurement takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |
| showHexValueText | Display the hex value of the selected color below the color chip. | 0 (don't show) | 0 or 1 |
| showRGBValueText | Display the RGB value of the selected color below the color chip. | 0 (don't show) | 0 or 1 |
| default_text | Default color value. | | Hex color value (Ex. FFCC33) |
| allowDuplicateValues | Allow duplicate values to be attached to a record. | 0 (don't allow duplicates) | 0 or 1 |

### Attribute settings: File size

Accepts digital file size values with commonly used suffixes: B, KB, KiB, MB, MiB, GB, GiB, TB, Tib, PB and PiB. Available from version 1.7.9.

| Settings | Description | Default | Values |
|---|---|---|---|
| fieldWidth | Width, in characters, of the field when displayed in a user interface. | 40 | Integers greater then zero |
| fieldHeight | Height, in characters, of the field when displayed in a user interface. | 1 | Integers greater then zero |
| requireValue | Use this option if you want an error to be thrown if this measurement is left blank. | 0 (can be blank) | 0 or 1 |
| doesNotTakeLocale | Defines whether measurement takes locale specification. | 1 (does not take locale specifications) | 0 or 1 |
| canBeUsedInSort | Use this option if this attribute value can be used for sorting of search results. | 1 (used in sort) | 0 or 1 |
| canBeUsedInSearchForm | Use this option if the attribute value can be used in search forms. | 1 (used in search forms) | 0 or 1 |
| canBeUsedInDisplay | Use this option if the attribute value can be used for display in search results. | 1 (used for display) | 0 or 1 |
| displayTemplate | Layout for value when used in a display. Element code tags prefixed with the ^ character, used to represent the value in the template. For example: <i>^my_element_code</i>. | | HTML |
| displayDelimiter | Delimiter to use between multiple values. | , (comma) | Text |
| allowDuplicateValues | Allow duplicate values to be attached to a record. | 0 (don't allow duplicates) | 0 or 1 |

### 1.9.2 Information Services

CollectiveAccess supports several external information services to attached metadata to CollectiveAccess records. It does this by performing a lookup operation at the remote service and then allowing you to pick a value from a results list. It stores core information about the referenced piece of data and a reference (URI) to the original resource. To configure metadata fields in the user interface, select the InformationService or LCSH *metadata element type*.

InformationService is also a plugin API that makes it easy to add support for other external services. The exact information stored locally differs from plugin to plugin.

- *Library of Congress Plugins (LC)*
- *Example LC installation profile code*
- *Information Services Plugins*
- *Example Information Service installation profile code*
- *Implementing new plugins*

### Library of Congress Plugins (LC)

- LC subject headings
- LC Name Authority file
- LC subject headings for children
- LC Genre/Forms File
- Thesaurus of Graphic Materials
- Preservation Events
- Preservation Level Role
- Cryptographic Hash Functions
- MARC Relators
- MARC Countries
- MARC Geographic Areas
- MARC Languages
- ISO639-1 Languages
- ISO639-2 Languages
- ISO639-5 Languages

### Example LC installation profile code

```
<metadataElement code="lcsh_terms" datatype="LCSH">
        <labels>
 <label locale="en_US">
          <name>Library of Congress Subject Headings</name>
   <description>Library of Congress Subject headings describing this object.</
↪description>
 </label>
        </labels>
<settings>
   <setting name="fieldWidth">80</setting>
  <setting name="fieldHeight">1</setting>
  <setting name="vocabulary">cs:http://id.loc.gov/authorities/subjects</setting>
        </settings>
        <typeRestrictions>
        <restriction code="ca_objects">
        <table>ca_objects</table>
                <settings>
                <setting name="minAttributesPerRow">0</setting>
                <setting name="maxAttributesPerRow">100</setting>
                <setting name="minimumAttributeBundlesToDisplay">1</setting>
        </settings>
        </restriction>
</typeRestrictions>
</metadataElement>
```

### Information Services Plugins

#### Getty Vocabularies

AAT (Art and Architecture Thesaurus), TGN (Thesaurus of Geographic Names), and ULAN (The Union List of Artist Names) are all provided via SparqlEndpoint Linked Open Data Service by the Getty Vocabularies. None of these 3 plugins has any custom settings on element level, but they share a more comprehensive configuration in the configuration file *Linked_data.conf*. The default configuration should work for most use cases.

#### ALA-National Species List

"Open access to the Atlas of Living Australia's biodiversity data"

#### CollectiveAccess

This plugin allows you to reference records in remote CollectiveAccess instances. Available settings are as follows:

Table 1: CollectiveAccess Information Service

| Setting Name | Description | Example |
|---|---|---|
| service | Set service setting to 'CollectiveAccess' to use this plugin | CollectiveAccess |
| baseURL | URL used to query the information service | http://localhost/admin/ |
| table | valid CollectiveAccess table name | ca_entities |
| user_name | User name to authenticate with on remote system | webservice |
| password | Password to authenticate with on remote system | / |
| labelFormat | Display template to format query result labels | ^ca_entities.preferred_labels |
| detailFormat | Display template to format detailed information blocks | ^ca_objects.preferred_labels (^ca_objects.idno) |

#### Encyclopedia of Life (EOL)

"Global acccess to knowledge about life on Earth"

#### Iconclass

"A multilingual classification system for cultural content"

#### ResourceSpace

"ResourceSpace is a web-based Digital Asset Management software offering a solution for organising and sharing files"

### VIAF

"Open access to linked names for the same entity across the world's major name authority files, including national and regional variations in language, character set, and spelling."

### Wikipedia

This service allows referencing Wikipedia articles. Available settings are:

Table 2: Wikipedia Information Service Installation Profile Settings

| Setting Name | Description | Example |
|---|---|---|
| service | Set service setting to 'Wikipedia' to use this plugin | Wikipedia |
| lang | 2- or 3-letter language code for Wikipedia to use. Defaults to en | en |

### Wikipedia Display Template Options

This plugin can pull in data for local display. For example, both the abstract and preview image are available in bundle displays. Suppose your wikipedia metadata element has the code **wikipedia**. You can reference additional properties about a referenced article like this:

```
ca_objects.wikipedia.<property>
```

Where property is one of the following:

Table 3: Wikipedia Information Service Installation Profile Settings

| Setting Name | Description |
|---|---|
| image_thumbnail | Image thumbnail URL |
| image_thumbnail_width | Width of image thumbnail. Box is capped at 200px by 200px |
| image_thumbnail_height | Height of image thumbnail. Box is capped at 200px by 200px |
| image_viewer_url | (Valid for v1.5.1) URL for Wikipedia's full screen image viewer |
| title | Title of the Wikipedia article |
| pageid | Numeric page identifier |
| fullurl | URL for the article |
| canonicalurl | Canonical URL for the article |
| extract | Extract of the article, usually a HTML representation of the full article |
| abstract | CollectiveAccess tries to extract the first paragraph from the article to provide a shorter abstract. This is usually the part shown above the table of contents but extraction might fail for poorly formatted articles |

### WorldCat

To use WorldCat you'll need either a valid OCLC Z39.50 login or WorldCat Web Search API key. The two connection method have different technical requirements but offer identical functionality. Note that your OCLC user agreement may prohibit you from using the Web Search API for cataloguing activity. Consult your OCLC service representative as to your rights before using the API. Your PHP installation must have cURL support to use the Web Search API. PHP must be built with YAZ support to use Z39.50. YAZ is available as a standard package on many Linux distributions and installation is generally straightforward.

Specify your Web Search API key or Z39.50 login in App.conf. The entries are:

- worldcat_api_key
- worldcat_z39.50_user
- worldcat_z39.50_password

### WorldCat Options

> You can use WorldCat as a metadata element type "information service", but you can also use it to import bibliographic data from WorldCat directly into your CollectiveAccess system using the WorldCat import interface, available in the "Import" menu. The importer works much as the general data importer, but with a specialized interface for interactively locating and retrieving one or more entries from WorldCat. For more information, read the <PLACE LINK HERE> Importer documentation.

### Example Information Service installation profile code

```
    <metadataElement code="my_element" datatype="InformationService">
  <labels>
    <label locale="en_US">
      <name>My InformationService Element</name>
    </label>
  </labels>
  <settings>
    <setting name="service"><!-- enter service here --></setting>
  </settings>
  <typeRestrictions>
    <restriction code="r1">
      <table>ca_objects</table>
      <settings>
        <setting name="minAttributesPerRow">0</setting>
        <setting name="maxAttributesPerRow">255</setting>
        <setting name="minimumAttributeBundlesToDisplay">1</setting>
      </settings>
    </restriction>
  </typeRestrictions>
</metadataElement>
```

### Implementing new plugins

InformationService implementations reside in *app/lib/core/Plugins/InformationService* and should implement IWLPlugInformationService and extend BaseInformationServicePlugin. The class name must be "WLPlugInformationService<Service>" and the file name "<Service>.php".

It can provide additional settings using the static $s_settings variable, usually derived from $g_information_service_settings_<Service>. It should set the "NAME" property of the info array in the constructor.

The Wikipedia implementation is relatively simple and uses most of the available features (except getDataForSearchIndexing()) so you could use that as a template.

**Core functions**

The core functions you must implement are:

```
public function lookup($pa_settings, $ps_search, $pa_options=null);
```

where $pa_settings is an array containing the settings for this particular element (including the ones you provided) and $ps_search is the search expression provided by the user. The function should return an array with the "results" key being a list results for the given search expression. Each result should have a label, url and idno.

```
public function getExtendedInformation($pa_settings, $ps_url);
```

This should return an array with the "display" key set to an HTML representation of the given record (identified by the URL/URI). You can either go and look the detailed data up remotely or, for instance, call getExtraInfo() to get locally stored data (see below).

**Optional functions**

The functions listed below are optional and have default (empty) implementations in BaseInformationServicePlugin so it doesn't hurt to leave them out of your plugin entirely. They can be used to provide useful features though.

```
public function getExtraInfo($pa_settings, $ps_url);
```

Returns an array of key=>value pairs containing extra information to be stored locally, alongside the id, the display label and the URL. This data can be accessed using SearchResult::get(), so you should keep the keys alphanumeric, lowercase and without spaces.

```
public function getDataForSearchIndexing($pa_settings, $ps_url);
```

Returns a list of strings that are added to the search index for the record associated with this attribute. This allows you to add additional data points that can be used to find the CollectiveAccess record but are not necessarily available for display. Note that the data returnd by getExtraInfo() is not indexed for search, so you might have to add the same data twice.

```
public function getDisplayValueFromLookupText($ps_text);
```

The default behavior is to use the (selected) label returned by the lookup() function as display value for attribute values. That can be undesirable for use cases like the AAT where one the one hand you want a lot of identifying information in the lookup dropdown but on the other you probably don't care about all that info once the "relationship" has been created because the keyword is doing its job in the background (making the associated record findable). Maybe you just want a simple and short label instead to save space.

This function allows you to mangle the lookup text to create a different display value. The lookup text usually has the URL in it, so you could even look up additional info to pull in here if you wanted. An example can be found in the AAT implementation, where we do some regular expression magic to convert lookup texts:

```
before: [300025342] swordsmiths [people in crafts and trades by product, people in␣
↪crafts and trades]
after: swordsmiths
```

## 1.9.3 Measurements

CollectiveAccess can

## 1.9.4 Date and Time Formats

CollectiveAccess can process dates and times in a variety of formats. Internally, CA represents date/times as a range with a beginning and an end. This means that your dates can be as precise or imprecise as necessary. For example, 2007, June 2007, June 7 to June 10 2007 and June 7 2007 are all valid dates. They are stored internally as

- January 1 2007 @ 00:00:00am - December 31 2007 @ 12:59:59pm
- June 1 2007 @ 00:00:00am - June 30 2007 @ 12:59:59pm
- June 7 2007 @ 00:00:00am - June 10 2007 @ 12:59:59pm
- June 7 2007 @ 00:00:00am - June 7 2007 @ 12:59:59pm

respectively.

The actual low-level storage format represents dates as a pair of numbers representing the start and end of a date range rather than text. This allows CA to properly search and sort dates no matter what format and language is used to enter them. It also allows CA to impose a standard display format on dates regardless of their original input format and to re-format and translate dates on the fly without requiring changes to your underlying data.

### Languages and Localization

Date/time expressions are output in the users' current locale language. Language specific settings are defined in TimeExpressionParser locale configuration files stored in app/lib/core/Parsers/TimeExpressionParser

### Configuration

It is possible to configure how dates and times are parsed and displayed using the datetime.conf configuration file

### Valid input formats

| | | |
|---|---|---|
| Year | 2016, 1950 ad; 450 b.c.; 40 mya | simple years as well as AD and BC dates and geologic time (mya aka "millions of years ago") are supported |
| Month and year | June 2016, 6/2016 | |
| Specific date | June 6 2016; June 7, 2016; 6/7/2016; 6/7/16 | Support for European style dates (eg. day first rather than month first) is based upon users' current locale setting |

For numeric dates (eg. 6/7/2007) multiple delimiters are supported. For example, in the US localization the following dates are all valid and equivalent:

6/7/2007

6-7-2007

6.7.2007

7-JUN-2007

7-JUN-07

### Dates with times

You can specify a time for any date by following the date with a time expression. Both 24 hour and 12 hour (AM/PM) times are supported, and you can specify times to the minute or second. For readability you can optionally separate the date and time with a separator. For the US localization allowable separators are:

at, @

| | | |
|---|---|---|
| Date with 24 hour time | June 7, 2007 16:43; 6/7/2007 @ 16:43; June 7 2007 at 16:43 | Specified to the minute: 4:43pm |
| Date with 12 hour time | June 7, 2007 4:43:03pm; 6/7/2007 @ 4:43:03p.m. | Time specified to the second |

If you omit the time then a time is assumed depending upon whether the date is the beginning, end or both of a date range. For dates at the beginning of a range, the default time is 00:00:00 (midnight). For dates at the end of a range the default time is 11:59:59pm. This means that if you input a date without a time the entire day is encompassed.

The elements of a time specification may be delimited in multiple ways. For the US localization the following delimiters are supported:

```
:
.
```

Thus the following times are valid and equivalent:

4.15:05pm

16.15.05

4:15:05pm

16:15:05

You can also enter date/times in ISO 8601 format. Note that CollectiveAccess has no provision for recording time zones. All times are assumed to be in the same time zone and any time zone information in ISO-format dates is currently discarded (this may change in a future release).

### Date ranges

You can specify a date range by inputting two dates (with or without times) separated by a range separator. For the US localization, the range separators are:

to, -, and, .., through

For readability you can also include an optional range indicator before the first date. For the US localization range indicators are:

from, between

Examples of date ranges:

June 5, 2007 - June 15, 2007

Between June 5, 2007 and June 15 2007

From 6/5/2007 to 6/15/2007

6/5/2007 @ 9am .. 6/5/2007 @ 5pm

6/5 .. 6/15/2007 (Note implicit year in first date)

6/5 at 9am - 5pm (Note implicit date in current year with range of times)

### Unbounded dates

Date ranges where one end is unspecified can be expressed in various. ways. Ranges with a specified start date but no end date are considered to be ongoing and can be expressed in any of the following (using the example start date June 6 1944):

6/6/1944 to present

6/6/1944 - present

6/6/1944 .. present

after 6/6/1944

6/6/1944 -

6/6/1944 - ?

Date ranges where the end date is specified and the start date unspecified are considered to include ‘’any” date prior to the end date. They may be specified using the formats:

before 6/6/1944

? - 6/6/1944

### Special expressions

There are a number of shorthand expressions for common dates. Examples below are for the English localization, but all localizations support them:

today (current date to the day)

yesterday (yesterday’s date to the day)

tomorrow (tomorrow’s date to the day)

now (current date/time to the second)

1990’s (decade)

199- (AACR2 format decade)

20th century (century)

19– (AACR2 format century)

### Early/mid/late dates

As of version 1.7.7 it is possible to qualify decade and century dates and date ranges with “early”, “mid” and “late” modifiers. CollectiveAccess will interpret “early” centuries expressions as being between the start of the century and the 21st year. Eg. “Early 18th Century” will be stored as 1 January 1700 - 31 December 1720. “Late” dates are considered to be between the 81st year and the end of the century. Eg “Late 18th Century” will be stored as 1 January 1780 - 31 December 1799. “Mid 18th Century” will be stored as 1 January 1740 - 31 December 1760. For decades are treated similarly: “Early 1920s” is stored as 1 January 1920 - 31 December 1923. “Mid 1920s” is stored as 1 January 1923 to 31 December 1927. “Late 1920s” is stored as 1 January 1926 to 31 December 1929.

The rules for mapping early, mid and late ranges to concrete dates are current built into the parser and cannot be changed. They may be made configurable in future versions.

### Uncertain dates

You can express uncertain dates in two ways:

Preface the date with a “circa” specifier (in English, use “circa”, “c” or “ca”). Add a question mark (“?”) to the end of the date

For example:

circa 1955

ca June 1865

May 2 1921?

As of version 1.1 you can also use “circa” with date ranges:

circa 1950 - 1956

### Imprecise dates

“Circa” indicates merely that the date is not precisely known. It does not convey an information about the margin of error of the date estimate. If you want to specify a numeric margin of error for a date/time us expressions such as these:

June 10 1955 ~ 10d (June 10th 1955 plus or minus 10 days)

1955 ~ 3y (1955 plus or minus 3 years)

### Eras

All dates are assumed to be in the Common Era (CE) unless otherwise specified. In the English localization you can specify a date before the Common Era by appending “BCE”:

850 BCE

You may also append “CE” for common era dates if you wish. The English localization also supports use of “AD” and “BC” Other localizations may use different modifiers.

### Year-less dates

It is possible to enter dates that lack years if needed. Year-less dates are restricted to delimited date format input and are available at the month and month/day level:

6/10/????

6/????

Note that any number of question marks will create a valid date/time.

### Seasonal dates

As of version 1.1 of CollectiveAccess, seasonal dates are supported. Simply enter the name of the season optionally followed by a year (the current year is assumed if there is no year input), and CollectiveAccess will convert the date to numbers. In the English locale, valid seasonal input might include:

Summer 2011

Fall 2009

These expressons map to specific dates, June 21 2011 to September 20 2011 for Summer for example.

### Quarter-century dates

Ranges of years falling on quarter centuries may be input as century/quarter pairs. For example:

> 20 Q3

is equivalent to 1950 - 1975 (3rd quarter of 20th century). Quarter century expressions are always in the Common Era. They cannot be used for BC dates.

### Undated

You may indicate a date-less item using "undated" or "unknown" (in the standard English translation, at least). "Undated" date expressions imply the absence of date, and are not searchable. They exists only to indicate that no date is known.

Attribute types represent all of the different varieties of fields supported by CollectiveAccess. Each one is slightly different because each is designed to accept and normalized data in different formats. See below for more details about the unique nature of each Attribute type. The full list of attribute types is listed here.

| Type code | Description |
|---|---|
| Container | Unlike all other attribute types, containers do not represent data values. Rather their sole function is to organ |
| Text | Represents a free-text value. |
| DateRange | Represents an historic date range accepting date/time range expressions in the formats acceptedby the CA Ti |
| List | Represents a value chosen from a drop-down list populated with values from a specified list as defined in the |
| Geocode | Represents one or more latitude/longitude coordinates. Coordinates can be entered as decimal latitude/longit |
| Url | Accepts a properly formatted URL value. Currently does fairly limited checking on overall syntax and speci |
| Currency | Accepts a currency value composed of a currency specifier and a decimal number. The current specifier shou |
| Length | Accepts length measurements in metric, English and points units (the latter being typographical points). Ent |
| Weight | Accepts weight measurements in metric and English units. Entries are simply a numeric quantity + a unit sp |
| TimeCode | Accepts time offsets in a number of time code formats. Time code values are typically used to express a dura |
| Integer | Accepts integer values, but no floats. In effect everything that contains only the digits 0-9 is accepted. |
| Numeric | Accepts numeric values. Numeric strings consist of optional sign, any number of digits, optional decimal pa |
| LCSH | Library of Congress Subject Heading value. Takes LCSH heading or first part of heading (the LCSH search |
| GeoNames | GeoNames value. Takes search text, passes it to the GeoNames search service and provides a dropdown wit |
| File | Uploaded file. CA will try to identify the file and extract limited metadata, but even if it can't identify the fil |
| Media | Uploaded media (image, sound video). CA will try to identify the file, extract metadata and create derivative |
| Taxonomy | Taxonomic name as returned from a taxonomic name service. Currently support lookups into ITISand uBio |
| InformationService | Remote webservice lookup. Lookup values web services including the Getty TGN, ULAN and AAT, anothe |
| ObjectRepresentations | Reference object representations within your CollectiveAccess instance. |
| Entities | Entity value. Searches the CollectiveAccess entity authority for given text and creates a typeless pseudo-rela |
| Places | Place value. Searches the CollectiveAccess place authority for given text and creates a typeless pseudo-relat |
| Occurrences | Occurrence value. Searches the CollectiveAccess occurrence authority for given text and creates a typeless p |
| Collections | Collection value. Searches the CollectiveAccess collection authority for given text and creates a typeless pse |
| StorageLocations | Storage Location value. Searches the CollectiveAccess storage location authority for given text and creates a |
| Loans | Loan value. Searches the CollectiveAccess loan authority for given text and creates a typeless pseudo-relatio |
| Movements | Movement value. Searches the CollectiveAccess movement authority for given text and creates a typeless ps |
| Objects | Object value. Searches the CollectiveAccess object authority for given text and creates a typeless pseudo-rel |
| ObjectLots | Object Lot value. Searches the CollectiveAccess object lot authority for given text and creates a typeless pse |
| Floorplan | Implements an interactive floorplan user interface for place authority records |
| Color | Represents a color in standard RGB hex format (Ex. FFCC33). In the editing interface this data type is typic |
| Filesize | Accepts digital file size values with commonly used suffixes (B, KB, KiB, MB, MiB, GB, GiB, TB, TiB, PB |

## 1.10 Relationships

To come

## 1.11 Interstitial Data

As of version 1.4, CollectiveAccess supports relationship records also known as "interstitial" records. This feature allows cataloguers to describe a relationship beyond simply selecting a relationship type. Let's say, for example, that you have two entities that were married for a period of time, and then got divorced. The relationship record allows you to add a date range, narrative text and/or other metadata elements of your choosing to the interstice between these two individuals. Relationship records are entirely optional, and in fact won't be accessible unless a user interface is defined for the them. Relationship records are not just limited to entities. Any two records can carry this interstitial description, so long as metadata and a user interface has been created. Other common examples of relationships that could require interstitial metadata include objects to places; objects to entities; entities to places, etc.

### 1.11.1 Setting up relationship records in the installation profile

To create a metadata element with an interstitial type restriction in the profile requires adopting some of the syntax used for relationshipTable names. Here's how you would add the date range on an entity to entity relationship record:

```
<metadataElement code="relationshipDate" datatype="DateRange">
      <labels>
        <label locale="en_US">
          <name>Relationship date</name>
          <description/>
        </label>
      </labels>
     <settings/>
      <typeRestrictions>
        <restriction code="r1">
          <table>ca_entities_x_entities</table>
          <settings>
            <setting name="minAttributesPerRow">1</setting>
            <setting name="maxAttributesPerRow">1</setting>
            <setting name="minimumAttributeBundlesToDisplay">1</setting>
          </settings>
        </restriction>
      </typeRestrictions>
    </metadataElement>
```

After you've defined the metadata elements for your relationship record, you need to create the user interface. This follows the same syntax as the user interfaces for the main tables, except that the user interface "type" is the same string used in the typeRestriction "table" above:

```
<userInterface code="interstitial_entity_ui" type="ca_entities_x_entities">
      <labels>
        <label locale="en_US">
          <name>Interstitial Entity Editor</name>
        </label>
      </labels>
      <screens>
        <screen idno="basic" default="1">
          <labels>
            <label locale="en_US">
              <name>Basic info</name>
            </label>
          </labels>
          <bundlePlacements>
            <placement code="ca_attribute_relationshipDate">
              <bundle>ca_attribute_relationshipDate</bundle>
            </placement>
          </bundlePlacements>
        </screen>
      </screens>
    </userInterface>
```

Note that these interstitial records are meant to be small and manageable, so only one screen per user interface is supported. If other screens are defined they simply won't appear.

## 1.11.2 Setting up relationship records through the graphical user interface

Setting up a relationship record through the GUI is essentially just like creating a user interface for any other type of record. It follows the same steps wherein a metadata element is created and then added to the user interface.

The key difference is what "Type restrictions" are chosen for the elements and what "type" is used to create the user interface.

IMAGE:

The above metadata element will be used for an entity to entity relationship record.

IMAGE: New UI interface.png

The above user interface (created under Manage > Administration > User Interfaces) will be used for an object to storage location relationship record.

## 1.11.3 Editing Relationship Records

Once your metadata elements and user interface editors have been configured, you will notice a small edit icon on relevant relationships after they've been save the first time:

IMAGE: Paperclip.png

Clicking on the "E" will open the relationship record in an overlay.

## 1.12 Lists & Authorities

### 1.12.1 List item intrinsics (ca_list_items)

| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| Identifier | idno | The list item identifier. Must follow policy defined in configured numbering policy if app.conf setting require_valid_id_number_for_ca_list_items is set. Must be unique if app.conf setting allow_duplicate_id_number_for_ca_list_items is not set. | Depends upon numbering policy | |
| Type | type_id | A value from the list_item_types list indicating the type of the record. Stored as an internally generated numeric item_id. When setting this value in a data import or via an API call the item identifier may be used. | Yes | null |
| Parent | parent_id | Reference to parent record. Will be null if no parent is defined. When setting this value in a data import or via an API call the identifier of the parent place may be used. | No | null |
| List | list_id | A reference to the list record (ca_lists) of which the list item is a part. Note that a list item is always part of a list. It cannot exist outside of a list. The raw database value contained list_id is an internally generated numeric list_id. However, when setting this intrinsic via an import mapping or API call you may also use the list's code. | No | |
| Value | item_value | Value represented by list item. This is distinct from the identifier and used to convey a text or numeric quantity when required. | Yes | |
| Access | access | Determines visibility of record in public-facing applications such as Pawtucket. Values are defined in the access_statuses list. Typically the list includes values for "public" and "private" visibility. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. By convention "0" is interpreted as private and "1" as public access, although this can be modified or expanded in app.conf if required. | Yes | 0 |
| Status | status | Records the general cataloguing workflow status of the record. Values are defined in the workflow_statuses list. For historical reasons the value stored in the intrinsic is the list item's *value* field, not its identifer or label. Unlike access values, statuses have no functional impact on a record. They are merely informations and intended to provide a simple, straightforward way to track the cataloguing process. | Yes | 0 |
| Icon | icon | Icon image to display for listitem. | No | |
| Color | color | Highlight color for list item in hex format. | No | |
| Is enabled? | is_enabled | Flag indicating whether list item is available for use (value set to 1) or not available (value is 0). | Yes | 0 |
| Is default? | is_default | Flag indicating whether list item is the default selection for its list (value set to 1) or not | Yes | 0 |

### 1.12.2 List intrinsics (ca_lists)

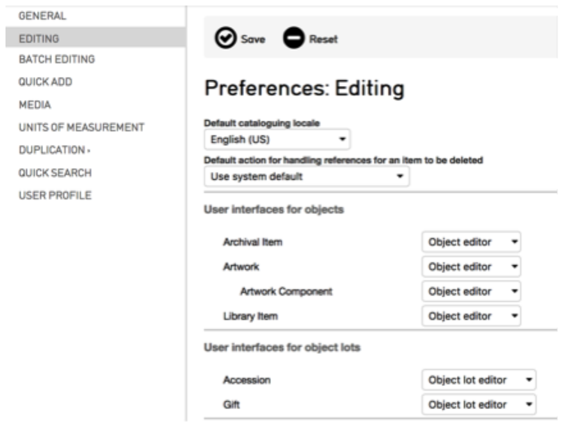
| Name | Code | Description | Mandatory? | Default |
|---|---|---|---|---|
| List code | list_code | The list identifier. Must be a unique alphanumeric code without spaces or punctuation beyond underscores. | Yes | |
| Is system list? | is_system_list | Flag indicating whether list is required to support application functionality (value set to 1) or is purely for cataloguing purposes (value is 0). | Yes | 0 |
| Is hierarchical? | is_hierarchical | Flag indicating whether list is hierarchical (value set to 1) or flat (value is 0). | Yes | 0 |
| Use as vocabulary? | use_as_vocabulary | Flag indicating whether list should be used in vocabulary (keyword) lookups (value set to 1) or not (value is 0). | Yes | 0 |
| Default sort | default_sort | Specifies the default method to employ to order items in this list. Takes a numeric value indicating the sort order: 0 = Sort by list item label 1 = Sort by list item rank 2 = Sort by list item value 3 = Sort by list item identifier | Yes | 0 |

## 1.13 User Interfaces

## 1.14 User Interface Administration

### 1.14.1 User Interface Configuration

There are several configuration options that can be set system wide or for individual users that can improve cataloging efficiency for editing interfaces. These configuration options are set via *Manage* > *Administration*> *User Interfaces*. If your account does not have access to administration configuration please contact your system administrator. Individual users can select the User Interface they prefer in *Manage* > *My Preferences* > *Editing*.



Preferences: Editing

Default cataloguing locale
English (US)

Default action for handling references for an item to be deleted
Use system default

User interfaces for objects

| | |
|---|---|
| Archival Item | Object editor |
| Artwork | Object editor |
| Artwork Component | Object editor |
| Library Item | Object editor |

User interfaces for object lots

| | |
|---|---|
| Accession | Object lot editor |
| Gift | Object lot editor |

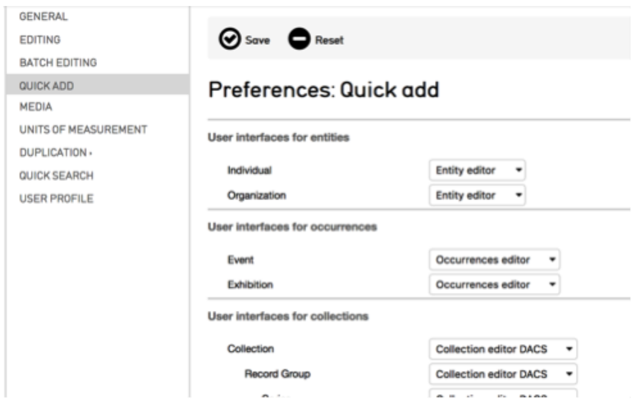
*Fig 5.1a: Selecting preferred user interface for record types in Manage > My Preferences > Editing*

**Creating User Interfaces**

It is possible to make multiple Editor Screen Interfaces for record types that are geared to specific cataloging tasks or users' working style (see the manual entry for Editor Screens). For example, if a cataloger is responsible for entering data in only a small subset of metadata elements for a record, a new User Interface can be configured with only those elements to support that specific workflow.

When creating User Interfaces, access to User Interfaces and User Interface screens can be set at the account, group and role level to limit availability to subsets of users (see the manual entry for Access Control).

Similarly, users can select User Interfaces for use in Quick Add forms in *Manage > My Preferences > Quick Add*. Creating custom User Interfaces for Quick Add screens can be very useful for ensuring a core subset of information is entered during the Quick Add process. See the manual entry for Relationships for more information on Quick Add.



Preferences: Quick add

User interfaces for entities

| | |
|---|---|
| Individual | Entity editor |
| Organization | Entity editor |

User interfaces for occurrences

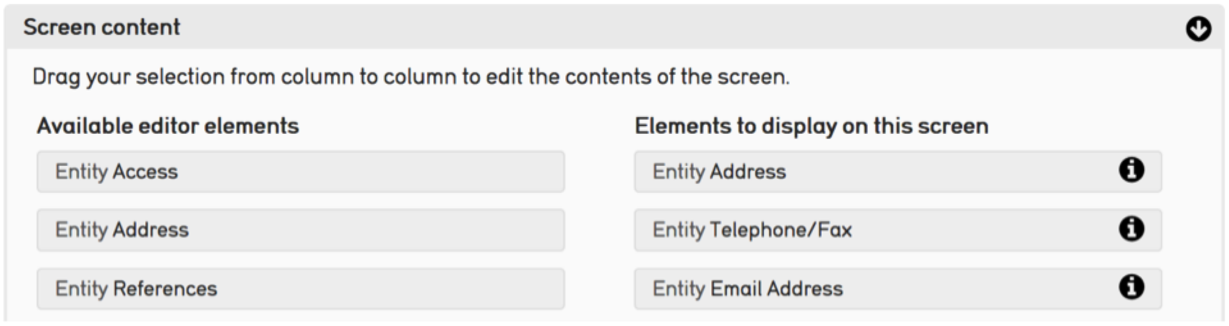
| | |
|---|---|
| Event | Occurrences editor |
| Exhibition | Occurrences editor |

User interfaces for collections

| | |
|---|---|
| Collection | Collection editor DACS |
| Record Group | Collection editor DACS |

Fig. 1: *Fig 5.1b: Manage > My Preferences > Quick Add*

**Configuring Metadata Elements in User Interfaces** To adjust the configuration of a metadata element within a User Interface, first edit a User Interface and select the screen the metadata element appears on. Access metadata element configuration options by clicking the information icon alongside element names in the list of "Elements to display on this screen". Settings are specific to the placement of the element within the current User Interface and will not apply to all placements of the element throughout the system.



Screen content

Drag your selection from column to column to edit the contents of the screen.

| Available editor elements | Elements to display on this screen |
|---|---|
| Entity Access | Entity Address |
| Entity Address | Entity Telephone/Fax |
| Entity References | Entity Email Address |

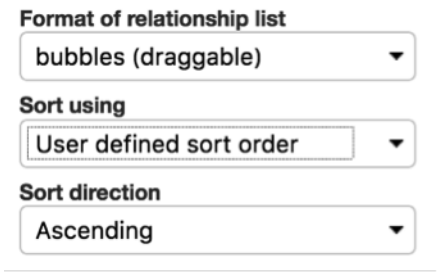
Fig. 2: *Fig 5.2: Selecting and configuring metadata elements in Manage > Administration > User Interfaces. Clicking the information icon reveals configuration options for elements within the User Interface screen.*

The following selection of options may be useful for customizing complex User Interfaces.

**Expand/ Collapse Metadata Elements**: Metadata elements within all editors can be expanded and collapsed by clicking the arrow icon to the right of each element's label. Users can configure metadata elements to be expanded or collapsed by default when forms are loaded. In the information window for metadata elements in a User Interface you will find two options, "Expand collapse if value exists" and "Expand collapse if no value is present". Both settings have the following options: Don't force (default), Collapse, Expand.

**Sort Elements in Repeating Bundles**: Sort order and direction for repeatable elements can be configured by selecting options from the "Sort using" and "Sort direction" dropdowns. Different sort options are available for different metadata elements.

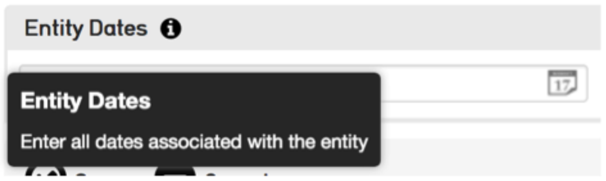
Metadata Elements for related records can be dragged and dropped into arbitrary order when the "Format of relationship list" dropdown is set to "Bubbles (draggable)" and "Sort using" is set to "User defined sort order".



Format of relationship list: bubbles (draggable)
Sort using: User defined sort order
Sort direction: Ascending

Fig. 3: *Fig 5.3: Configuring sort order and format of relationship lists in Manage > Administration > User Interfaces*

Configure Help Text and Documentation Links for Metadata Elements: Help text and links to documentation for metadata elements can be configured for elements or for elements within the context of a specific User Interface. Help text appears when users hover over the element title and links to documentation are available when clicking the information icon alongside the title of an element.



Entity Dates

Entity Dates
Enter all dates associated with the entity

Fig. 4: *Fig 5.4: Element help text revealed on hover. Information icon linked to URL in "Documentation URL" for the element*

Text entered in the "Description" field and links entered in the "Documentation URL" field for metadata elements in *Manage > Administration > Metadata Elements* are available wherever the element is used in a User Interface. Text entered in the "Descriptive text for bundle" and "Documentation URL" fields in the configuration options for elements within a User Interface screen in *Manage > Administration > User Interfaces* will only appear for that specific User Interface and will override the Metadata Element description and documentation URL.

## 1.15 Data Dictionary

To come

## 1.16 Locales

CollectiveAccess supports localizations of its user interface. It also supports multi-lingual metadata - you can add information translated into many languages to a object or authority record. Whenever a language must be specified, either to indicate how the UI should be localized or what language metadata is in, a locale code is used.

Locale codes are comprised of a two or three character country code taken from the ISO 639 standard and a two character language code taken from the ISO 3166-1 standard joined together by an underscore ("_") character. For example, English as spoken in the United States is indicated by **en_US**. German as spoken in Germany is indicated by **de_DE**. An optional dialect specifier may also be added following the country and separated with an underscore when sub-divisions of languages must be taken into account. (Note: the dialect specification may be dropped in the future; if you think this should stay let us know.)

**Locale-specific information in CA**

Locale-related information is stored into three places within CA:

- The list of locales for which CA can accept cataloguing is stored in the *ca_locales* table in the database. *ca_locales&* records simply contain the language, country and dialect codes along with a unique, compact integer identifier that is used internally to represent the locale.
- Localized text for the user interface is stored in GNU GetText -compatible .po files in the *app/locale/<locale_code>* directory (for example: *app/locale/en_US*). Note that not all locales present in the *ca_locales* table will have localization files in *app/locale*, but that all locales *'with'* localization files will have a corresponding record in *ca_locales*.
- Configuration specifying how date/time expressions are parsed and presented for a given locale are stored in *app/lib/core/Parsers/TimeExpressionParser* When creating a new locale it is best to clone an existing locale file and alter it to suit the new locale.
- CA uses Zend_Locale (from the Zend Framework) to manage other localization data (number formats, country and language names and the like). The data is stored in *app/lib/core/Zend/Locale/Data* in files using standard locale codes.

**How locales are selected**

Each CA user may select a preferred locale for their user interface; the list of locales is limited to those for which a user interface translation exists in *app/locale*. The selected locale is used not only for the user interface but also, when possible, for displayed content. That is, CA will always try to display catalogued data in the user's preferred UI locale. If catalogue data is not available in the user's selected language then CA will attempt to display the data using one of the system default locales, as set in the global.conf configuration file. The default locales will be used in order of listing until catalogue data is found. If no catalogue data exists for the default locales, the CA will display data in whatever locale *does* exist.

**Translating CA into a new language**

For information on how to translate CA into a language for which a translation does not yet exist see the the page on creating Translations.

## 1.17 Labels

To come

## 1.18 Configuring Providence

Providence has many configuration options beyond the initial setup through installation profiles. These options are handle in .conf files in the **app/conf** directory of your Providence installation. You will also find a **/local** folder, where you should create blank copies of files to customize options.

**Attention:** ALWAYS create a blank local version of your .conf file in app/conf/local and make edits there. Otherwise your changes can be overwritten.

### 1.18.1 app.conf

General configuration of CollectiveAccess is controlled by the *app.conf* configuration file.

## Structure

To make editing app.conf a bit more manageable the file is broken up into several sections grouping related options: menus, disable switches, hierarchies, titles and identifiers, search, features, access control, styling, mapping, defaults, media, administration and esoterica (things you probably won't ever need to change). For sections where a separate configuration file is also available (Eg. search.conf and the search section) the app.conf section controls only high-level options, while the file handles detailed configuration.

WHAT: This is the main application configuration file for Providence, designed so that users can easy manage various system-wide settings in one convenient location.

WHEN TO CUSTOMIZE: App.conf handles most customizations that are not controlled through the UI or profile (with the notable exceptions of: browse, id number configurations, and additional settings for dates, media and search). This document is broken into the following sections: Disable, Hierarchies, Titles & Ids, Search, Feature Settings, Access Control, Styling, Mapping, System Defaults, Media, Admin Configuration and Export.

## Menus

Control the "New" and "Find" menus in Providence

### Menu bar preferences

By default each of the follow record types has a sub-menu in the top-level "New" menu list out the configured types. When you choose a type the creation of new record of that type is initiated. If you have several types configured sub-menus make sense, but if your setup only has one or two types, or is deeply nested then you may want to push the first level of types directly onto the menu. Setting the directives below will force the first level of the sub-menu onto the "new" menu itself.

```
ca_object_lots_no_new_submenu = 0
ca_objects_no_new_submenu = 0
ca_entities_no_new_submenu = 0
ca_collections_no_new_submenu = 0
ca_loans_no_new_submenu = 0
ca_movements_no_new_submenu = 0
ca_tours_no_new_submenu = 0
ca_object_representations_no_new_submenu = 0
```

### Find menu formatting

By default only the top-level item classes ("objects", "entities", "collections") appear in the find menu Set the following to a non-zero value to break out each top-level item into a submenu with types This allows you do to type-specific searches and browses

```
ca_objects_breakout_find_by_type_in_submenu = 0
ca_object_lots_breakout_find_by_type_in_submenu = 0
ca_object_representations_breakout_find_by_type_in_submenu = 0
ca_entities_breakout_find_by_type_in_submenu = 0
ca_places_breakout_find_by_type_in_submenu = 0
ca_occurrences_breakout_find_by_type_in_submenu = 0
ca_collections_breakout_find_by_type_in_submenu = 0
ca_loans_breakout_find_by_type_in_submenu = 0
ca_movements_breakout_find_by_type_in_submenu = 0
ca_sets_breakout_find_by_type_in_submenu = 1
```

Set the following to a non-zero value to put types directly into the find menu, replacing the top-level item class This allows you do to type-specific searches and browses where the types are treated as generic top-level items

```
ca_objects_breakout_find_by_type_in_menu = 0
ca_object_lots_breakout_find_by_type_in_menu = 0
ca_object_representations_breakout_find_by_type_in_menu = 0
ca_entities_breakout_find_by_type_in_menu = 0
ca_places_breakout_find_by_type_in_menu = 0
ca_occurrences_breakout_find_by_type_in_menu = 1
ca_collections_breakout_find_by_type_in_menu = 0
ca_loans_breakout_find_by_type_in_menu = 0
ca_movements_breakout_find_by_type_in_menu = 0
ca_sets_breakout_find_by_type_in_menu = 0
```

### Navigation options

If you only want to allow users to create new records with top-level types for a give item type, set the appropriate directive below to 1; if set users will still be able to create records for sub-types, but only from within an existing record with a top-level types This can be useful if you have a system where sub-types need to be subsidiary to top-level records - eg. sub-type records need to have a top-level parent and cannot exist on their own

```
ca_objects_navigation_new_menu_shows_top_level_types_only = 0
ca_entities_navigation_new_menu_shows_top_level_types_only = 0
ca_places_navigation_new_menu_shows_top_level_types_only = 0
ca_occurrences_navigation_new_menu_shows_top_level_types_only = 0
ca_collections_navigation_new_menu_shows_top_level_types_only = 0
ca_object_lots_navigation_new_menu_shows_top_level_types_only = 0
ca_storage_locations_navigation_new_menu_shows_top_level_types_only = 0
ca_loans_navigation_new_menu_shows_top_level_types_only = 0
ca_movements_navigation_new_menu_shows_top_level_types_only = 0
ca_object_representations_navigation_new_menu_shows_top_level_types_only = 0
```

You can enumerate the types and sub-types shown in the new menu below.

```
ca_objects_navigation_new_menu_limit_types_to = []
ca_entities_navigation_new_menu_limit_types_to = []
ca_places_navigation_new_menu_limit_types_to = []
ca_occurrences_navigation_new_menu_limit_types_to = []
ca_collections_navigation_new_menu_limit_types_to = []
ca_object_lots_navigation_new_menu_limit_types_to = []
ca_storage_locations_navigation_new_menu_limit_types_to = []
ca_loans_navigation_new_menu_limit_types_to = []
ca_movements_navigation_new_menu_limit_types_to = []
ca_object_representations_navigation_new_menu_limit_types_to = []
```

### Show/Hide Representations

Sometimes you want representations enabled for relationship purposes but don't want to have the option to create or edit them as free-standing records. You can control whether the object representations, when enabled in general above, show up in the "new" and "find" menus using the directives below. Set them to a non-zero value to remove object representations from the specified menu.

```
ca_object_representations_dont_show_in_new_menu = 0
ca_object_representations_dont_show_in_find_menu = 0
```

### Show/Hide Tables

If you don't want certain modules to show up in the "New" menu, you can disable them here. They will still be searchable and can be created using QuickAdd or direct links (e.g. in the editor inspector of a related record, like an Object created from a Lot)

```
ca_objects_dont_show_in_new_menu = 0
ca_entities_dont_show_in_new_menu = 0
ca_places_dont_show_in_new_menu = 0
ca_occurrences_dont_show_in_new_menu = 0
ca_collections_dont_show_in_new_menu = 0
ca_object_lots_dont_show_in_new_menu = 0
ca_storage_locations_dont_show_in_new_menu = 0
ca_loans_dont_show_in_new_menu = 0
ca_movements_dont_show_in_new_menu = 0
```

### Menu bar caching

Caching the menu bar can significantly increase performance If you are developing a profile. caching can prevent you from seeing profile changes in real-time, however. So you can disable it here if need be. When using the system "in production" it is usually best to leave this enabled

```
do_menu_bar_caching = 0
```

### Disable

Turn off (or on) various features and database areas.

### Editor "disable" switches

If you're not using certain editors in your system (you don't catalogue places for example) you can disable the menu items for them by setting the various *_disable directives below to a non-zero value

```
ca_objects_disable = 0
ca_entities_disable = 0
ca_places_disable = 0
ca_occurrences_disable = 0
ca_collections_disable = 0
ca_object_lots_disable = 0
ca_storage_locations_disable = 0
ca_loans_disable = 0
ca_movements_disable = 1
ca_tours_disable = 1
ca_tour_stops_disable = 1
ca_object_representations_disable = 1
```

### QuickAdd disable switches

```
ca_objects_disable_quickadd = 0
ca_entities_disable_quickadd = 0
ca_places_disable_quickadd = 0
ca_occurrences_disable_quickadd = 0
ca_collections_disable_quickadd = 0
ca_object_lots_disable_quickadd = 0
ca_storage_locations_disable_quickadd = 0
ca_loans_disable_quickadd = 0
ca_movements_disable_quickadd = 0
```

### Disable "Add new <object> to lot"

(in the object lot editor inspector)

```
disable_add_object_to_lot_inspector_controls = 0
```

### Show related counts in the inspector?

```
ca_objects_show_related_counts_in_inspector_for = []
ca_entities_show_related_counts_in_inspector_for = [ca_objects]
ca_places_show_related_counts_in_inspector_for = []
ca_occurrences_show_related_counts_in_inspector_for = [ca_objects]
ca_collections_show_related_counts_in_inspector_for = [ca_objects]
ca_storage_locations_show_related_counts_in_inspector_for = []
ca_loans_show_related_counts_in_inspector_for = []
ca_movements_show_related_counts_in_inspector_for = []
ca_tour_stops_show_related_counts_in_inspector_for = []
```

### Show "add child record" control in editor inspector?

```
ca_objects_show_add_child_control_in_inspector = 0
ca_entities_show_add_child_control_in_inspector = 0
ca_places_show_add_child_control_in_inspector = 1
ca_occurrences_show_add_child_control_in_inspector = 0
ca_collections_show_add_child_control_in_inspector = 1
ca_storage_locations_show_add_child_control_in_inspector = 1
ca_loans_show_add_child_control_in_inspector = 0
ca_movements_show_add_child_control_in_inspector = 0
ca_tour_stops_show_add_child_control_in_inspector = 0
```

### Set duplication disable

If you want to disable the ability to duplicate all items in a set across the board set this

```
ca_sets_disable_duplication_of_items = 0
```

### Set type controls

```
enable_set_type_controls = 0
```

### Hierarchies

Settings for hierarchical properties and display.

### Strict type hierarchies

When fully enabled, top-level records may only be created with top-level types, and sub-records may only be created with types that are direct sub-types of the parent's type. This ensures conformance with the type hierarchy. So if you have an object type hierarchy like this:

**Book**

  **Page**

    **Figure** Frontspiece

. . . then top-level records can only be of type “Book.” Sub-records of books may only be “Page” or “Frontspiece”; and sub-records of “Page” can be “Figure.” “Frontspiece” may not take sub-records.

We partially enabled, top-level records may only be created with top-level types, but sub-records may be of *any* type below the top-level type, not just direct sub-types. In the example above, the sub-records of a “book” can be of type “Page”, “Figure” or Frontspiece; sub-records of a “Page” may be only of type “Figure.”

When disabled, all types are allowed anywhere.

The type hierarchy behavior can be independently for each type of hierarchical record. Set to 1 to fully enable, 0 to disable and ~ (tilde character) to partially enable restrictions.

```
ca_objects_enforce_strict_type_hierarchy = 0
ca_entities_enforce_strict_type_hierarchy = 0
ca_places_enforce_strict_type_hierarchy = 0
ca_occurrences_enforce_strict_type_hierarchy = 0
ca_collections_enforce_strict_type_hierarchy = 0
ca_storage_locations_enforce_strict_type_hierarchy = 0
ca_loans_enforce_strict_type_hierarchy = 0
ca_tour_stops_enforce_strict_type_hierarchy = 0
ca_list_items_enforce_strict_type_hierarchy = 0
```

### Hierarchy browser items

```
ca_objects_hierarchy_browser_display_settings = ^ca_objects.preferred_labels.name (^
↪ca_objects.idno)
ca_object_lots_hierarchy_browser_display_settings = ^ca_object_lots.preferred_labels␣
↪(^ca_object_lots.idno_stub)
ca_entities_hierarchy_browser_display_settings = ^ca_entities.preferred_labels (^ca_
↪entities.idno)
ca_places_hierarchy_browser_display_settings = ^ca_places.preferred_labels (^ca_
↪places.idno)
ca_occurrences_hierarchy_browser_display_settings = ^ca_occurrences.preferred_labels␣
↪(^ca_occurrences.idno)
ca_collections_hierarchy_browser_display_settings = ^ca_collections.preferred_labels␣
↪(^ca_collections.idno)
ca_list_hierarchy_browser_display_settings = ^ca_lists.preferred_labels.name (^ca_
↪lists.list_code)
ca_list_items_hierarchy_browser_display_settings = ^ca_list_items.preferred_labels.
↪name_plural (^ca_list_items.idno)
ca_storage_locations_hierarchy_browser_display_settings = ^ca_storage_locations.
↪preferred_labels (^ca_storage_locations.idno)
ca_tour_stops_hierarchy_browser_display_settings = ^ca_tour_stops.preferred_labels (^
↪ca_tour_stops.idno)
ca_relationship_types_hierarchy_browser_display_settings = ^ca_relationship_types.
↪preferred_labels (^ca_relationship_types.type_code)
ca_loans_hierarchy_browser_display_settings = ^ca_loans.preferred_labels (^ca_loans.
↪idno)
ca_movements_hierarchy_browser_display_settings = ^ca_movements.preferred_labels (^ca_
↪movements.idno)
```

```
ca_objects_hierarchy_browser_sort_values = [ca_objects.idno_sort]
ca_objects_hierarchy_browser_sort_direction = asc
ca_object_lots_hierarchy_browser_sort_values = [ca_object_lots.idno_stub_sort]
ca_object_lots_hierarchy_browser_sort_direction = asc
ca_entities_hierarchy_browser_sort_values = [ca_entities.preferred_labels.surname, ca_
↪entities.preferred_labels.forename]
```

(continues on next page)

(continued from previous page)

```
ca_entities_hierarchy_browser_sort_direction = asc
ca_places_hierarchy_browser_sort_values = [ca_places.rank, ca_places.preferred_labels.
↪name_sort]
ca_places_hierarchy_browser_sort_direction = asc
ca_occurrences_hierarchy_browser_sort_values = [ca_occurrences.preferred_labels.name_
↪sort]
ca_occurrences_hierarchy_browser_sort_direction = asc
ca_collections_hierarchy_browser_sort_values = [ca_collections.rank, ca_collections.
↪preferred_labels.name_sort]
ca_collections_hierarchy_browser_sort_direction = asc
ca_list_items_hierarchy_browser_sort_values = [ca_list_items.preferred_labels.name_
↪sort_plural]
ca_list_items_hierarchy_browser_sort_direction = asc
ca_list_items_hierarchy_browser_disabled_items_mode = disabled
ca_storage_locations_hierarchy_browser_sort_values = [ca_storage_locations.rank, ca_
↪storage_locations.preferred_labels.name_sort]
ca_storage_locations_hierarchy_browser_sort_direction = asc
ca_storage_locations_hierarchy_browser_disabled_items_mode = disabled
ca_tour_stops_hierarchy_browser_sort_values = [ca_tour_stops.preferred_labels.name_
↪sort]
ca_tour_stops_hierarchy_browser_sort_direction = asc
ca_relationship_types_hierarchy_browser_sort_values = [ca_relationship_types.
↪preferred_labels.typename]
ca_relationship_types_hierarchy_browser_sort_direction = asc
ca_loans_hierarchy_browser_sort_values = [ca_loans.preferred_labels.name_sort]
ca_loans_hierarchy_browser_sort_direction = asc
ca_movements_hierarchy_browser_sort_values = [ca_movements.preferred_labels.name_sort]
ca_movements_hierarchy_browser_sort_direction = asc
```

### Collection hierarchies on the Summary screen

The summary screen includes a visual hierarchy by default for hierarchical collections. Use these directives to set the sort value for the hierarchical display, as well as the display template used for format data. If nothing is set below the system will default to the settings outlined in ca_collections_hierarchy_browser_sort_values.

```
ca_collections_hierarchy_summary_display_settings =
ca_collections_hierarchy_summary_sort_values =
ca_objects_hierarchy_summary_display_settings =
ca_collections_hierarchy_summary_show_full_object_hierarachy = 0
```

### Show/Hide hierarchy root (Storage Locations & Places)

Hide hierarchy root for storage locations or places in New and Find screens Note that if you haven't added any items to the hierarchies yet, enabling this might prevent you from doing so (because you can't select a parent).

```
ca_storage_locations_hierarchy_browser_hide_root = 0
ca_places_locations_hierarchy_browser_hide_root = 0
```

### Show/Hide child records in search/browse results

Normally all results, regardless of their position in a hierarchy are displayed in search/browse results. Set this option for alternative policies. Possible settings are:

| Setting | Description | Allowed values | Required? | Default | Synonyms |
|---|---|---|---|---|---|
| <table>_children_display_mode_in_results | Normally all results, regardless of their position in a hierarchy are displayed in search/browse results. This option enables alternative policies, including permanent suppression and user-controlled filtering of child record. This option can be used with any primary table. | show = show all results by default; allow user to filter children if they wish hide = hide all child records (those that are not at the top of their hierarchy) by default; allow user to remove filtering if desired alwaysShow = show all results; do not allow filtering alwaysHide = hide all child records; do not allow the user to disable filtering | No | alwaysShow | |

"alwaysShow" is the default.

While this option may be set for any table, it is typically used only for objects.

```
ca_objects_children_display_mode_in_results = alwaysShow
```

### Enable display of collections and objects as a single hierarchy

```
ca_objects_x_collections_hierarchy_enabled = 1
ca_objects_x_collections_hierarchy_relationship_type =
ca_objects_x_collections_hierarchy_disable_object_collection_idno_inheritance =
```

### Titles + IDs

Set whether or not titles and identifiers are required and unique.

### Require input id number value to conform to format? (0=no, 1=yes)

```
require_valid_id_number_for_ca_objects = 0
require_valid_id_number_for_ca_object_lots = 0
require_valid_id_number_for_ca_entities = 1
require_valid_id_number_for_ca_places = 1
require_valid_id_number_for_ca_collections = 1
require_valid_id_number_for_ca_occurrences = 1
require_valid_id_number_for_ca_loans = 0
require_valid_id_number_for_ca_movements = 0
require_valid_id_number_for_ca_tours = 0
require_valid_id_number_for_ca_tour_stops = 0
require_valid_id_number_for_ca_object_representations = 0
require_valid_id_number_for_ca_storage_locations = 0
```

### Allow dupe id numbers? (0=no, 1=yes)

```
allow_duplicate_id_number_for_ca_objects = 1
allow_duplicate_id_number_for_ca_object_lots = 1
allow_duplicate_id_number_for_ca_entities = 1
allow_duplicate_id_number_for_ca_places = 1
```

(continues on next page)

(continued from previous page)

```
allow_duplicate_id_number_for_ca_collections= 1
allow_duplicate_id_number_for_ca_occurrences= 1
allow_duplicate_id_number_for_ca_list_items= 1
allow_duplicate_id_number_for_ca_loans= 0
allow_duplicate_id_number_for_ca_movements= 0
allow_duplicate_id_number_for_ca_tours= 0
allow_duplicate_id_number_for_ca_tour_stops= 0
allow_duplicate_id_number_for_ca_object_representations = 1
allow_duplicate_id_number_for_ca_storage_locations = 1
```

### Allow dupe labels? (0=no, 1=yes)

If set to no, then atttempting to save records with a label already in use by another record will fail

```
allow_duplicate_labels_for_ca_objects = 1
allow_duplicate_labels_for_ca_object_lots = 1
allow_duplicate_labels_for_ca_entities = 0
allow_duplicate_labels_for_ca_places = 1
allow_duplicate_labels_for_ca_collections= 0
allow_duplicate_labels_for_ca_occurrences= 0
allow_duplicate_labels_for_ca_storage_locations= 1
allow_duplicate_labels_for_ca_list_items= 1
allow_duplicate_labels_for_ca_loans = 1
allow_duplicate_labels_for_ca_movements= 1
allow_duplicate_labels_for_ca_object_representations= 1
allow_duplicate_labels_for_ca_relationship_types= 1
allow_duplicate_labels_for_ca_set_items= 1
allow_duplicate_labels_for_ca_search_forms= 1
allow_duplicate_labels_for_ca_bundle_displays= 1
allow_duplicate_labels_for_ca_metadata_alert_rules = 1
allow_duplicate_labels_for_ca_editor_uis= 1
allow_duplicate_labels_for_ca_editor_ui_screens= 1
allow_duplicate_labels_for_ca_tours= 1
allow_duplicate_labels_for_ca_tour_stops= 1
```

### Entity dupe name?

Set this to 1 if you want to display a warning when a new entity with a name that already exists (preferred or nonpreferred) is about to be created

```
ca_entities_warn_when_preferred_label_exists = 0
```

### Require preferred label? (0=no, 1=yes)

If set to yes, then attempting to save records without a preferred label will fail. If set to no (default) then attempting to save a record without a preferred label will automatically set the preferred label to "[BLANK]"

```
require_preferred_label_for_ca_objects = 0
require_preferred_label_for_ca_object_lots = 0
require_preferred_label_for_ca_entities = 0
require_preferred_label_for_ca_places = 0
```

(continues on next page)

(continued from previous page)

```
require_preferred_label_for_ca_collections = 0
require_preferred_label_for_ca_occurrences = 0
require_preferred_label_for_ca_storage_locations = 0
require_preferred_label_for_ca_list_items = 0
require_preferred_label_for_ca_loans = 0
require_preferred_label_for_ca_movements = 0
require_preferred_label_for_ca_object_representations = 0
require_preferred_label_for_ca_relationship_types = 0
require_preferred_label_for_ca_set_items = 0
require_preferred_label_for_ca_search_forms = 0
require_preferred_label_for_ca_bundle_displays = 0
require_preferred_label_for_ca_editor_uis = 0
require_preferred_label_for_ca_editor_ui_screens = 0
require_preferred_label_for_ca_tours = 0
require_preferred_label_for_ca_tour_stops = 0
```

### Require preferred label value be present in a list

If set to a valid list code then any entered label value must match a preferred label for an item in that list.

```
preferred_label_for_ca_objects_must_be_in_list =
preferred_label_for_ca_object_lots_must_be_in_list =
preferred_label_for_ca_entities_must_be_in_list =
preferred_label_for_ca_places_must_be_in_list =
preferred_label_for_ca_collections_must_be_in_list =
preferred_label_for_ca_occurrences_must_be_in_list =
preferred_label_for_ca_storage_locations_must_be_in_list =
preferred_label_for_ca_list_items_must_be_in_list =
preferred_label_for_ca_loans_must_be_in_list =
preferred_label_for_ca_movements_must_be_in_list =
preferred_label_for_ca_object_representations_must_be_in_list =
preferred_label_for_ca_relationship_types_must_be_in_list =
preferred_label_for_ca_set_items_must_be_in_list =
preferred_label_for_ca_search_forms_must_be_in_list =
preferred_label_for_ca_bundle_displays_must_be_in_list =
preferred_label_for_ca_editor_uis_must_be_in_list =
preferred_label_for_ca_editor_ui_screens_must_be_in_list =
preferred_label_for_ca_tours_must_be_in_list =
preferred_label_for_ca_tour_stops_must_be_in_list =
```

### Allow automated renumbering objects with lot idno + sequence number?

(when object number don't conform to that format)

If you're managing lots with related object-level records and the lot and object numbering get out of sync (because you change the lot number after the fact, for example) then this can be useful. But it can also be dangerous in the sense that letting cataloguers renumber sets of objects at a click may not be the idea. Only enable this if you need it. Keep in mind that the automated renumbering format is fixed at lot <lot identifier> + <separator> + <sequential number starting from one>. So if your lot number is 2010.10 and your separator is '.', then objects will be numbered 2010.10.1, 2010.10.2, 2010.10.3, etc.

```
allow_automated_renumbering_of_objects_in_a_lot = 0
```

#### Label-less objects

If you don't want to specify preferred labels for objects set this to a non-zero value This can be useful for collections where individual items lack working names, such as in paleontology.

```
ca_objects_dont_use_labels = 0
```

#### Label-specific sort

Set to assume a specific language when generating sortable titles regardless of the locale set for the title. This can be useful when content has been entered specific (or accurate) locale settings. The value can be a specific locale (Ex. "en_US") or a language code (Ex. "en")

```
use_locale_for_sortable_titles =
```

### Search

#### Search engine configuration

```
search_engine_plugin = SqlSearch
```

#### Browse Panel Styles

(for best results, choose a number between 1 and 5)

```
browse_row_size = 4
```

#### Quicksearch - order and results ("live" search in search box in header)

What sorts of results does Quicksearch return? List table names here to include them in the search, in the order they should appear. This is only the default display configuration, which can be overriden by user preferences. Syntax is ca_table/type, i.e ca_objects/video

```
quicksearch_default_results = [ca_objects, ca_entities, ca_places, ca_occurrences, ca_
↪collections, ca_object_lots, ca_storage_locations, ca_loans, ca_movements, ca_tours,
↪ ca_tour_stops]
```

#### Quicksearch - break out by type?

What table types are broken out in the result list? Syntax is list within square brackets, i.e [ca_objects, ca_entities]

```
quicksearch_breakout_by_type =
```

Restrict facets shown to specific facet groups?

```
<table_name>_browse_facet_group limits facets on the main browse landing page
<table_name>_refine_facet_group limits facets in the "refine" browse on detail pages
<table_name>_search_refine_facet_group limits facets in the "refine" browse on search␣
↪results
```

```
ca_objects_browse_facet_group = main
ca_objects_refine_facet_group = refine
ca_objects_search_refine_facet_group = refine
```

### One table search

If set to a controller in the “find” module, will use that for quicksearch rather than the regular “Quicksearch” controller. This is useful for having the Quicksearch operate on a single table

```
one_table_search =
```

### Out of process search indexing

Switch to disable out of process search indexing

```
disable_out_of_process_search_indexing = 0
```

Hostname to use when triggering out of process indexing By default the site hostname configured in setup.php is used but you can override it here if the hostname resolvable on the server differs from that used for incoming requests out_of_process_search_indexing_hostname =

### Caption formatting for search/browse “thumbnail” results

Set a display template here to customize display of captions under thumbnails in the thumbnail result view. The template will be evaluated relative to each record in the result set.

```
ca_objects_results_thumbnail_caption_template = ^ca_objects.preferred_labels.name
↪%truncate=27&ellipsis=1<br/><l>^ca_objects.idno</l>
ca_occurrences_results_thumbnail_caption_template = ^ca_occurrences.preferred_labels.
↪name%truncate=27&ellipsis=1<br/><l>^ca_occurrences.idno</l>
ca_entities_results_thumbnail_caption_template = ^ca_entities.preferred_labels.name
↪%truncate=27&ellipsis=1<br/><l>^ca_entities.idno</l>
ca_collections_results_thumbnail_caption_template = ^ca_collections.preferred_labels.
↪name%truncate=27&ellipsis=1<br/><l>^ca_collections.idno</l>
```

### Features

Settings related to various features such as: location tracking, deaccessioning, WorldCat, check in/check out and more.

### Location tracking options

Direct object-location reference storage location tracking (also set this for movement-based storage location tracking)

```
object_storage_location_tracking_relationship_type =
```

### Movement-based storage location tracking

```
movement_storage_location_tracking_relationship_type =
movement_object_tracking_relationship_type =
record_movement_information_when_moving_storage_location = 0
movement_storage_location_date_element =
```

### Deaccession options

```
deaccession_force_access_private = 1
deaccession_dont_allow_editing = 0
deaccession_use_disposal_date = 1
```

### Library-style check-out of objects

```
enable_library_services = 0
enable_object_checkout = 0
```

### User generated content

```
enable_user_generated_content = 1
```

### ResourceSpace import

The ResourceSpace data importer allows records and media to be imported from a ResourceSpace Installation The importer connects using a username and API Key that is unique to that user and can be found in the edit user page under the Admin > Manage Users tab in ResourceSapce

Also required is the base URL for your ResourceSpace installation which all API calls are based on This should be your root url + /api/

```
resourcespace_apis = {
    EXAMPLE_CARE_SYSTEM = {
        resourcespace_label = ,
        resourcespace_base_api_url = ,
        resourcespace_user =
    }
}
```

### WorldCat import

The data importer can access OCLC WorldCat via either their web service API or Z39.50 service. Using the web service API requires that PHP be installed with libCURL support. Using Z39.50 requires that PHP be built with libyaz support (http://www.indexdata.com/yaz). Many PHP installations have libCURL installed by default; most do not have libyaz installed.

The importer will connect ia Z39.50 if a username and password are configured below and libyaz is available, otherwise the web service API will be used as a fallback, assuming a valid API key is configured below and libCURL is available.

```
worldcat_api_key = MY_WORLDCAT_API_KEY
worldcat_z39.50_user =
worldcat_z39.50_password =
```

Optionally mark WorldCat items already present in system using ISBN To enable set "worlcat_isbn_element_code" to the ca_objects metadata element code containing the ISBN code for the book. worlcat_isbn_element_code =

Display template used to format "ISBN present" message. Evaluated relative to the existing object. You can use standard display template codes (eg. ^ca_objects.idno) to display details about the match.

```
worlcat_isbn_exists_template = <span class="caWorldCatExistingObjectIcon"><l><i class=
↪"fa fa-external-link" aria-hidden="true"></i></span></l>
```

Template formatting the "key" displayed below WorldCat query results. Use this to define any icons used in the "worlcat_isbn_exists_template"

```
worlcat_isbn_exists_key = <div class="caWorldCatExistingObjectKey"><i class="fa fa-
↪external-link" aria-hidden="true"></i> = Previously imported</div>
```

### Taxonomy web services

To access the uBio taxonomic name service (http://www.ubio.org) via a 'Taxonomy' attribute you must enter your uBio API keycode here If you don't care about taxonomy (or even know what is it) then leave this as-is

```
ubio_keycode = enter_your_keycode_here
```

### Flickr API

```
flickr_api_key =
```

### "Rich text" (aka. wysiwyg) editor options

You can read more about available text editor options here: http://docs.cksource.com/CKEditor_4.x/Developers_Guide/Toolbar

Defines options available in the toolbar

```
wysiwyg_editor_toolbar = {
    formatting = [Bold, Italic, Underline, Strike, -, Subscript, Superscript, Font,␣
↪FontSize, TextColor],
    lists = [-, NumberedList, BulletedList, Outdent, Indent, Blockquote],
    links = [Link, Unlink, Anchor],
    misc = [SelectAll, Undo, Redo, -, Source, Maximize, Image, CALink]
}
```

Defines options available in the toolbar

```
wysiwyg_content_editor_toolbar = {
    formatting = [Bold, Italic, Underline, Strike, -, Subscript, Superscript, Font,␣
↪FontSize, TextColor],
    lists = [-, NumberedList, BulletedList, Outdent, Indent, Blockquote],
    links = [Link, Unlink, Anchor],
    misc = [SelectAll, Undo, Redo, -, Source, Maximize, Media, CALink]
}
```

### Enable dependent field visibility feature

See here for more information: http://docs.collectiveaccess.org/wiki/Dependent_Field_Visibility

```
enable_dependent_field_visibility = 0
```

### Global template values (Pawtucket content management)

Globals are text values that may be set in the Pawtucket web UI and substituted into any view template, including headers and footers. Values defined here will be editable in the "Global Values Editor" (available to users with the can_edit_theme_global_values priv) and usable in templates under their name (Eg. {{{operating_hours}}} in the example below).

Options controlling how the editor displays the value may be set for each global. Width and height control the size of the element; usewysiwygeditor enables a "wysiwyg" rich text editor for formatted text.

```
global_template_values = {
    hours_of_operation = {
        name = Hours of operation,
        description = List current operating hours here,
        width = 600px,
        height = 150px,
        usewysiwygeditor = 0
    }
}
```

### Site page templates (Pawtucket content management)

Allow PHP code in content-managed site pages

By default only value tags in the form {{{tag-name}}} are allowed in Pawtucket site page templates. If you need the flexibility and power afforded by direct embedding of PHP code in your templates set this option to a non-zero value. Note that enabling this option will allow execution of ANY code embedded in the template on EVERY page load. Depending upon your point of view this is either a feature or a security hole. It doesn't have to be a problem, but keep it in mind. . .

Note that this setting only affects page previews in Providence. To allow PHP code execution in Pawtucket you must also set this option in your theme.

```
allow_php_in_site_page_templates = 0
```

### Access Control

Structural mechanisms that control who can see what, and how (optional).

#### Bundle-level access control

```
default_bundle_access_level = __CA_BUNDLE_ACCESS_EDIT__
```

#### Type-level access control

```
perform_type_access_checking = 0
default_type_access_level = __CA_BUNDLE_ACCESS_EDIT__
```

#### Source-level access control

```
perform_source_access_checking = 0
default_source_access_level = __CA_BUNDLE_ACCESS_EDIT__
```

#### Item-level access control

```
perform_item_level_access_checking = 0
default_item_access_level = __CA_ACL_EDIT_DELETE_ACCESS__
```

You can control item-level access control support for each type of item using these directives

```
ca_objects_dont_do_item_level_access_control = 0
ca_object_lots_dont_do_item_level_access_control = 0
ca_entities_dont_do_item_level_access_control = 0
ca_places_dont_do_item_level_access_control = 0
ca_occurrences_dont_do_item_level_access_control = 0
ca_collections_dont_do_item_level_access_control = 0
ca_lists_dont_do_item_level_access_control = 0
ca_list_items_dont_do_item_level_access_control = 0
ca_loans_dont_do_item_level_access_control = 0
ca_movements_dont_do_item_level_access_control = 0
ca_object_representations_dont_do_item_level_access_control = 0
ca_representation_annotations_dont_do_item_level_access_control = 0
ca_storage_locations_dont_do_item_level_access_control = 0
ca_tours_dont_do_item_level_access_control = 0
ca_tour_stops_dont_do_item_level_access_control = 0
```

**Defaults for collection-to-object ACL inheritance settings** Set to 1 to make default to inherit; 0 for default to be no inheritance

```
ca_collections_acl_inherit_from_parent_default = 0
ca_objects_acl_inherit_from_ca_collections_default = 0
ca_objects_acl_inherit_from_parent_default = 0
```

### Administrator

User_id to consider "administrator" - not subject to access control measures. By default, user_id=1 is considered administrator for convenience and compatbility with older installations. You can make any user_id "administrator" if you want, however, if disable this completely by setting it to a blank value.

```
administrator_user_id = 1
```

email user when account is activated in Manage > Access control?

```
email_user_when_account_activated = 0
```

### Set Access

If you want all users to see all sets regardless of ownership or access control set this to one (Yes, some people apparently want to do this)

```
ca_sets_all_users_see_all_sets = 0
```

### "Access" inheritance

Allows child records to receive the "access" field value of their immediate parent. This can be useful when you generally want child record access to mirror that of the parent, but with occasional cataloguer-defined exceptions

Currently only supported for ca_objects

```
ca_objects_allow_access_inheritance = 0
```

Default inheritance status for newly created ca_objects records

```
ca_objects_access_inheritance_default = 1
```

### Styling

Controls for visual elements such as logos, colors, etc. within the application and exported reports and labels

### Theme configuration

To display your logo in the menu bar, upload it to the graphics/logos/ folder in the Default theme directory and enter the filename below. For the best results, your logo must not exceed 45 pixels in height. To change the menu color, enter the six digit HTML color code below and omit the leading ‘’ sign.

```
header_img = menu_logo.png
menu_color = ffffff
footer_color = ffffff
login_logo = ca_logo.png
```

### Search Result Reporting configuration

To display your logo at the top of a PDF report, upload it to the graphics/logos/ folder in all themes directory and enter the filename below. To change the header color (report_color) and header text color (report_text_color), enter the six digit HTML color code below and omit the leading ‘’ sign.

```
report_header_enabled = 1
report_img = menu_logo.png
report_color = FFFFFF
report_text_color = 000000
```

The following options control what additional information can be printed on your PDF reports. Enter a non-zero value to include the following information.

```
report_show_timestamp = 1
report_show_number_results = 1
report_representation_version = preview
report_show_search_term = 1
```

### Record PDF Summary configuration

To display your logo at the top of a PDF report, upload it to the graphics/logos/ folder in all themes directory and enter the filename below. To change the header color (summary_color) and header text color (summary_text_color), enter the six digit HTML color code below and omit the leading ‘’ sign.

```
summary_header_enabled = 1
summary_page_numbers = 1
summary_footer_enabled = 1
summary_img = ca_wide.png
summary_color = FFFFFF
summary_text_color = 000000
summary_footer_color = FFFFFF
summary_footer_text_color = 000000
```

The following options control what additional information can be printed on your PDF summary. Enter a non-zero value to include the following information.

```
summary_show_identifier = 1
summary_show_timestamp = 1
```

/* Image path for icon to display when no image is available in thumbnail view */ /* Image must be uploaded to graphics/buttons in your theme folder */

```
no_image_icon = glyphicons_138_picture.png
```

### Print labels (ie. stickers)

As of CollectiveAccess version 1.5 a new label generator is available that is easier to configure and customize. The new generator uses HTML/CSS to specify the layout of label formats, unlike the old system which uses a set of complex configuration files. Any existing label formats you wish to use with the new generator must be completely reimplemented. There is no automated conversion process.

```
Set this if you want a dashed border around all printed labels
add_print_label_borders = 0
```

### Annotation options

element code of ca_representation_annotation list metadata element that should be used to classify and color code annotations

```
annotation_class_element =
```

### Additional theme

theme to use when user is not logged in (when they're logged in their preferred theme is used)

```
theme = default
themes_directory = <ca_base_dir>/themes
themes_url = <ca_url_root>/themes
views_directory = <themes_directory>/<theme>/views
```

### Mapping

Settings for GeoNames and Mapping plugins (Google Maps/Open Layers)

### GeoNames web services

To access the GeoNames services for geographic names via a 'GeoNames' attribute you must enter your GeoNames username and password here. You can get a free account at http://www.geonames.org/login. After you confirmed your registration you have to enable your account for web service usage at http://www.geonames.org/manageaccount, otherwise the search won't return any results. If you don't care about GeoNames (or even know what is it) then leave this as-is

```
geonames_user = enter_your_username_here
```

The api.geonames.org URL should not be changed if you're using the free GeoNames web service. The free offering should be sufficient for most users. If you have a paid/premium account, geonames provides you with a list of additional hostnames available for use over https here: http://www.geonames.org/account Enter one of those hostnames to make use of your premium subscription

```
geonames_api_base_url = http://api.geonames.org
```

### Mapping plugins

**Name of plugin class to use for mapping** Currently supported values: OpenLayers, Leaflet

OpenLayers is deprecated. Use Leaflet unless you have a reason to do otherwise. mapping_plugin = Leaflet

**Leaflet options** Show zoom in/out control `leaflet_maps_show_scale_controls = 1`

Path color for polygons and circles `leaflet_maps_path_color = "0000cc"`

Path weight (in pixels) for polygons and circles `leaflet_maps_path_weight = 2`

Path opacity for polygons and circles (0 is transparent, 1 is opaque) `leaflet_maps_path_opacity = 0.6`

Fill color for polygons and circles `leaflet_maps_fill_color = "ff0000"`

Fill opacity for polygons and circles (0 is transparent, 1 is opaque) `leaflet_maps_fill_opacity = 0.1`

URL for base layer when using Leaflet mapping plugin See https://leaflet-extras.github.io/leaflet-providers/preview/ for previews of various base maps

`leaflet_base_layer = https://maps.wikimedia.org/osm-intl/{z}/{x}/{y}{r}.png`

**OpenLayers options** Tile to use for base layer; Ex. OpenLayers.Layer.OSM() [OpenStreetMaps] or OpenLayers.Layer.Stamen('toner') [Stamen 'Toner' theme]

`openlayers_base_layer = OpenLayers.Layer.OSM()`

Radius, in pixels, of plotted points

`openlayers_point_radius = 5`

Fill color (hex) for points and regions

`openlayers_fill_color = ffcc66`

Stroke width, in pixels, for points, regions and paths

`openlayers_stroke_width = 2`

Stroke color (hex) for points, regions and paths

`openlayers_stroke_color = ff9933`

Fill color (hex) for points and regions when selected

`openlayers_fill_color_selected = 66ccff`

Stroke color (hex) for points regions and paths when selected

`openlayers_stroke_color_selected = 3399ff`

**Generic mapping options** Attribute object records to use to map search results

`ca_objects_map_attribute = georeference`

### Defaults

System defaults to control layouts, displays, templates and more.

### Related item lookup settings

```
ca_objects_lookup_settings = [<unit relativeTo='ca_objects'>^ca_object_
↪representations.media.icon (^ca_objects.idno) ^ca_objects.preferred_labels</unit>]
ca_objects_lookup_delimiter =
ca_objects_lookup_relationship_type_position = right
ca_objects_lookup_sort = _natural;ca_objects.idno_sort
ca_objects_lookup_relationship_type_editable = 0

ca_object_lots_lookup_settings = [^ca_object_lots.preferred_labels (^ca_object_lots.
↪idno_stub)]
ca_object_lots_lookup_delimiter =
```

(continues on next page)

(continued from previous page)

```
ca_object_lots_lookup_relationship_type_position = right
ca_object_lots_lookup_sort = _natural;ca_object_lots.idno_stub_sort
ca_object_lots_lookup_relationship_type_editable = 0

ca_entities_lookup_settings = [^ca_entities.preferred_labels]
ca_entities_lookup_delimiter =
ca_entities_lookup_relationship_type_position = right
ca_entities_lookup_sort = _natural;ca_entity_labels.name_sort
ca_entities_lookup_relationship_type_editable = 0

ca_places_lookup_settings =  [^ca_places.hierarchy.preferred_labels.name
↪%maxLevelsFromBottom=4]
ca_places_lookup_delimiter =
ca_places_lookup_relationship_type_position = right
ca_places_lookup_sort = _natural;ca_places.idno_sort
ca_places_lookup_relationship_type_editable = 0

ca_occurrences_lookup_settings = [^ca_occurrences.preferred_labels]
ca_occurrences_lookup_delimiter =
ca_occurrences_lookup_relationship_type_position = right
ca_occurrences_lookup_sort = _natural;ca_occurrences.idno_sort
ca_occurrences_lookup_relationship_type_editable = 0

ca_collections_lookup_settings = [^ca_collections.preferred_labels (^ca_collections.
↪idno)]
ca_collections_lookup_delimiter =
ca_collections_lookup_relationship_type_position = right
ca_collections_lookup_sort = _natural;ca_collections.idno_sort
ca_collections_lookup_relationship_type_editable = 0

ca_storage_locations_lookup_settings = [^ca_storage_locations.hierarchy.preferred_
↪labels.name]
ca_storage_locations_lookup_delimiter =
ca_storage_locations_lookup_relationship_type_position = right
ca_storage_locations_lookup_sort = _natural;ca_storage_locations.idno_sort
ca_storage_locations_lookup_relationship_type_editable = 0

ca_list_items_lookup_settings = [^ca_list_items.hierarchy.preferred_labels.name_
↪plural]
ca_list_items_lookup_delimiter =
ca_list_items_lookup_relationship_type_position = right
ca_list_items_lookup_sort = _natural;ca_list_items.idno_sort
ca_list_items_lookup_relationship_type_editable = 0

ca_relationship_types_lookup_settings = [^ca_relationship_types.parent.preferred_
↪labels  ^ca_relationship_types.preferred_labels (^ca_relationship_types.type_code)]
ca_relationship_types_lookup_delimiter =
ca_relationship_types_lookup_sort = _natural;ca_relationship_types.type_code

ca_loans_lookup_settings = [^ca_loans.preferred_labels]
ca_loans_lookup_delimiter =
ca_loans_lookup_relationship_type_position = right
ca_loans_lookup_sort = _natural;ca_loans.idno_sort
ca_loans_lookup_relationship_type_editable = 0

ca_movements_lookup_settings = [^ca_movements.preferred_labels]
ca_movements_lookup_delimiter =
```

(continues on next page)

(continued from previous page)

```
ca_movements_lookup_relationship_type_position = right
ca_movements_lookup_sort = _natural;ca_movements.idno_sort
ca_movements_lookup_relationship_type_editable = 0

ca_users_lookup_settings = [^ca_users.fname ^ca_users.lname (^ca_users.email)]
ca_users_lookup_delimiter =
ca_users_lookup_sort = _natural;ca_users.user_name

ca_user_groups_lookup_settings= [^ca_user_groups.name]
ca_user_groups_lookup_delimiter =
ca_user_groups_lookup_sort = _natural;ca_user_groups.code

ca_tours_lookup_settings = [^ca_tours.preferred_labels]
ca_tours_lookup_delimiter =
ca_tours_lookup_sort = _natural;ca_tours.tour_code

ca_tour_stops_lookup_settings = [^ca_tour_stops.preferred_labels]
ca_tour_stops_lookup_delimiter =
ca_tour_stops_lookup_sort = _natural;ca_tour_stops.idno_sort
ca_tour_stops_lookup_relationship_type_editable = 0

ca_object_representations_lookup_settings = [^ca_object_representations.media.icon ^
↪ca_object_representations.preferred_labels]
ca_object_representations_lookup_delimiter =
ca_object_representations_lookup_sort = _natural;ca_object_representations.idno_sort
ca_object_representations_lookup_relationship_type_editable = 0

ca_representation_annotations_lookup_settings = [^ca_representation_annotations.
↪preferred_labels.name]
ca_representation_annotations_lookup_delimiter =
ca_representation_annotations_lookup_sort = _natural

ca_sets_lookup_settings = [^ca_sets.preferred_labels.name (^ca_sets.set_code)]
ca_sets_lookup_delimiter =
ca_sets_lookup_sort = _natural

ca_object_checkouts_lookup_settings = [^ca_objects.preferred_labels.name (^ca_objects.
↪idno) <i>Borrowed on ^ca_object_checkouts.checkout_date%timeOmit=1 by ^ca_users.
↪fname ^ca_users.lname</i>]
ca_object_checkouts_lookup_delimiter =
```

### Default bundle display templates for related bundles (Eg. ca_entities, ca_occurrences, etc.)

```
ca_objects_default_bundle_display_template = <unit relativeTo="ca_objects"><l>^ca_
↪objects.preferred_labels.name</l> (^relationship_typename)</unit>
ca_entities_default_bundle_display_template = <unit relativeTo="ca_entities"><l>^ca_
↪entities.preferred_labels.displayname</l> (^relationship_typename)</unit>
ca_places_default_bundle_display_template = <unit relativeTo="ca_places"><l>^ca_
↪places.preferred_labels.name</l> (^relationship_typename)</unit>
ca_occurrences_default_bundle_display_template = <unit relativeTo="ca_occurrences"><l>
↪^ca_occurrences.preferred_labels.name</l> (^relationship_typename)</unit>
ca_object_lots_default_bundle_display_template = <unit relativeTo="ca_object_lots"><l>
↪^ca_object_lots.preferred_labels.name</l> (^ca_object_lots.idno_stub)</unit>
ca_storage_locations_default_bundle_display_template = <unit relativeTo="ca_storage_
↪locations"><l>^ca_storage_locations.preferred_labels.name</l> (^relationship_
↪typename)</unit>
```

(continues on next page)

(continued from previous page)

```
ca_loans_default_bundle_display_template = <unit relativeTo="ca_loans"><l>^ca_loans.
↪preferred_labels.name</l> (^relationship_typename)</unit>
ca_movements_default_bundle_display_template = <unit relativeTo="ca_movements"><l>^ca_
↪movements.preferred_labels.name</l> (^relationship_typename)</unit>
ca_object_representations_default_bundle_display_template = <unit relativeTo="ca_
↪object_representations" delimiter="<br/>"><l>^ca_object_representations.media.
↪thumbnail</l><br/><l>^ca_object_representations.preferred_labels.name</l> (^
↪relationship_typename)</unit>
ca_list_items_default_bundle_display_template = <unit relativeTo="ca_list_items"><l>^
↪ca_list_items.preferred_labels.name_plural</l> (^relationship_typename)</unit>
```

### Default template for media viewer caption

```
media_overlay_titlebar_template = "^ca_objects.preferred_labels.name <ifdef code='ca_
↪objects.idno'>(^ca_objects.idno)</ifdef>"
```

### Label type lists

Labels, both preferred and non-preferred, for primary items (objects, entities, etc.) can include a type. By default the range of types is defined by a list named for the item. For objects, the types for preferred labels are object_label_types_preferred while the non-preferred label types are defined by the object_label_types list. You can set other lists for each kind of label below. If you don't want to use types for a given category of label set it to an empty list.

```
ca_objects_preferred_label_type_list = object_label_types_preferred
ca_objects_nonpreferred_label_type_list = object_label_types
ca_object_lots_preferred_label_type_list = object_lot_label_types_preferred
ca_object_lots_nonpreferred_label_type_list = object_lot_label_types
ca_entities_preferred_label_type_list = entity_label_types_preferred
ca_entities_nonpreferred_label_type_list = entity_label_types
ca_places_preferred_label_type_list = place_label_types_preferred
ca_places_nonpreferred_label_type_list = place_label_types
ca_collections_preferred_label_type_list = collection_label_types_preferred
ca_collections_nonpreferred_label_type_list = collection_label_types
ca_occurrences_preferred_label_type_list = occurrence_label_types_preferred
ca_occurrences_nonpreferred_label_type_list = occurrence_label_types
ca_loans_preferred_label_type_list = loan_label_types_preferred
ca_loans_nonpreferred_label_type_list = loan_label_types
ca_movements_preferred_label_type_list = movement_label_types_preferred
ca_movements_nonpreferred_label_type_list = movement_label_types
ca_storage_locations_preferred_label_type_list = storage_location_label_types_
↪preferred
ca_storage_locations_nonpreferred_label_type_list = storage_location_label_types
ca_list_items_preferred_label_type_list = list_item_label_types_preferred
ca_list_items_nonpreferred_label_type_list = list_item_label_types
ca_object_representations_preferred_label_type_list = object_representation_label_
↪types_preferred
ca_object_representations_nonpreferred_label_type_list = object_representation_label_
↪types
ca_representation_annotation_preferred_label_type_list = representation_annotation_
↪label_types_preferred
ca_representation_annotation_nonpreferred_label_type_list = representation_annotation_
↪label_types
```

### Default to summary when opening item for editing?

```
ca_objects_editor_defaults_to_summary_view = 0
ca_object_lots_editor_defaults_to_summary_view = 0
ca_entities_editor_defaults_to_summary_view = 0
ca_places_editor_defaults_to_summary_view = 0
ca_occurrences_editor_defaults_to_summary_view = 0
ca_collections_editor_defaults_to_summary_view = 0
ca_lists_editor_defaults_to_summary_view = 0
ca_list_items_editor_defaults_to_summary_view = 0
ca_loans_editor_defaults_to_summary_view = 0
ca_movements_editor_defaults_to_summary_view = 0
ca_storage_locations_editor_defaults_to_summary_view = 0
ca_object_representations_editor_defaults_to_summary_view = 0
ca_tours_editor_defaults_to_summary_view = 0
ca_tour_stops_editor_defaults_to_summary_view = 0
ca_representation_annotations_defaults_to_summary_view = 0
```

### Find defaults

```
items_per_page_options_for_ca_objects_search = [12,24,36,48]
items_per_page_default_for_ca_objects_search = 24
view_default_for_ca_objects_search = list

items_per_page_options_for_ca_object_lots_search = [15,30,45]
items_per_page_default_for_ca_object_lots_search = 30
view_default_for_ca_object_lots_search = list
enable_full_thumbnail_result_views_for_ca_object_lots_search = 0

items_per_page_options_for_ca_entities_search = [15,30,45]
items_per_page_default_for_ca_entities_search = 30
view_default_for_ca_entities_search = list
enable_full_thumbnail_result_views_for_ca_entities_search = 0

items_per_page_options_for_ca_places_search = [15,30,45]
items_per_page_default_for_ca_places_search = 30
view_default_for_ca_places_search = list

items_per_page_options_for_ca_occurrences_search = [15,30,45]
items_per_page_default_for_ca_occurrences_search = 30
view_default_for_ca_occurrences_search = list
enable_full_thumbnail_result_views_for_ca_occurrences_search = 0

items_per_page_options_for_ca_collections_search = [15,30,45]
items_per_page_default_for_ca_collections_search = 30
view_default_for_ca_collections_search = list
enable_full_thumbnail_result_views_for_ca_collections_search = 0

items_per_page_options_for_ca_storage_locations_search = [15,30,45]
items_per_page_default_for_ca_storage_locations_search = 30
view_default_for_ca_storage_locations_search = list

items_per_page_options_for_ca_objects_browse = [12,24,36,48]
items_per_page_default_for_ca_objects_browse = 24
view_default_for_ca_objects_browse = list
```

(continues on next page)

(continued from previous page)

```
items_per_page_options_for_ca_object_lots_browse = [15,30,45]
items_per_page_default_for_ca_object_lots_browse = 30
view_default_for_ca_object_lots_browse = list
enable_full_thumbnail_result_views_for_ca_object_lots_browse = 0

items_per_page_options_for_ca_entities_browse = [15,30,45]
items_per_page_default_for_ca_entities_browse = 30
view_default_for_ca_entities_browse = list
enable_full_thumbnail_result_views_for_ca_entities_browse = 0

items_per_page_options_for_ca_places_browse = [15,30,45]
items_per_page_default_for_ca_places_browse = 30
view_default_for_ca_places_browse = list

items_per_page_options_for_ca_occurrences_browse = [15,30,45]
items_per_page_default_for_ca_occurrences_browse = 30
view_default_for_ca_occurrences_browse = list
enable_full_thumbnail_result_views_for_ca_occurrences_browse = 0

items_per_page_options_for_ca_collections_browse = [15,30,45]
items_per_page_default_for_ca_collections_browse = 30
view_default_for_ca_collections_browse = list
enable_full_thumbnail_result_views_for_ca_collections_browse = 0

items_per_page_options_for_ca_storage_locations_browse = [15,30,45]
items_per_page_default_for_ca_storage_locations_browse = 30
view_default_for_ca_storage_locations_browse = list

items_per_page_options_for_ca_loans_browse = [15,30,45]
items_per_page_default_for_ca_loans_browse = 30
view_default_for_ca_loans_browse = list

items_per_page_options_for_ca_movements_browse = [15,30,45]
items_per_page_default_for_ca_movements_browse = 30
view_default_for_ca_movements_browse = list

items_per_page_options_for_ca_lists_browse = [15,30,45]
items_per_page_default_for_ca_lists_browse = 30
view_default_for_ca_lists_browse = list

items_per_page_options_for_ca_list_items_browse = [15,30,45]
items_per_page_default_for_ca_list_items_browse = 30
view_default_for_ca_list_items_browse = list

items_per_page_options_for_ca_tours_browse = [15,30,45]
items_per_page_default_for_ca_tours_browse = 30
view_default_for_ca_tours_browse = list

items_per_page_options_for_ca_tour_stops_browse = [15,30,45]
items_per_page_default_for_ca_tour_stops_browse = 30
view_default_for_ca_tour_stops_browse = list

items_per_page_options_for_ca_object_representations_browse = [15,30,45]
items_per_page_default_for_ca_object_representations_browse = 30
view_default_for_ca_object_representations_browse = list
```

### Set item display templates

Used to format records in set item lists when no specific formatting has been specified

```
ca_objects_set_item_display_template = ^ca_objects.preferred_labels.name (^ca_objects.
↪idno)
ca_object_lots_set_item_display_template = ^ca_object_lots.preferred_labels.name (^ca_
↪object_lots.idno_stub)
ca_entities_set_item_display_template = ^ca_entities.preferred_labels.displayname
ca_places_set_item_display_template = ^ca_places.preferred_labels.name
ca_occurrences_set_item_display_template = ^ca_occurrences.preferred_labels.name
ca_collections_set_item_display_template = ^ca_collections.preferred_labels.name
ca_loans_set_item_display_template = ^ca_loans.preferred_labels.name
ca_movements_set_item_display_template = ^ca_movements.preferred_labels.name
ca_storage_locations_set_item_display_template = ^ca_storage_locations.preferred_
↪labels.name
ca_object_representations_set_item_display_template = ^ca_object_representations.
↪preferred_labels.name
ca_list_items_set_item_display_template = ^ca_list_itmes.preferred_labels.name_plural␣
↪(^ca_list_items.idno)
ca_tours_set_item_display_template = ^ca_tours.preferred_labels.name
ca_tour_stops_set_item_display_template = ^ca_tour_stops.preferred_labels.name
```

enable this to always show a default bundle preview for attribute bundles, even if the display template for that particular element isn't set

```
always_show_bundle_preview_for_attributes = 0
```

### Default type to use when creating sets

(in search results "sets" options, for example)

```
ca_sets_default_type = user
```

### Timecode output

Controls how timecode values are displayed Valid settings are:

- COLON_DELIMITED = format with colons. Ex. 1:20:10
- HOURS_MINUTES_SECONDS = format with h/m/s labels. Ex. 1h 20m 10s
- RAW = the number of seconds in the interval. Ex. 4810

```
timecode_output_format = COLON_DELIMITED
```

### Currency settings

By default currency values using the "$" symbol are considered to be in US dollars. You can change that here to another currency using its standard 3-letter code. Ex. CDN = Canadian dollars

```
default_dollar_currency = USD
```

### Length settings

Use Unicode fraction glyphs such as (ex. ¼) in place of the text equivalent (ex. 1/4)

As of version 1.7.6 these settings are DEPRECATED. In a future version these settings will be removed. Use the settings in the dimensions.conf configuration file if possible.

```
use_unicode_fractions_for_measurements = 1
force_use_of_fractions_for_measurements = 0
```

### Record duplication

By default duplicated records have the word "duplicate" appended to their preferred label. You can disable this behavior by setting this option.

```
dont_mark_duplicated_records_in_preferred_label = 0
```

### Log options

By default a timestamp is shown for every change in the record-based change log. Enable this to limit the display to the date of the change.

```
dont_show_timestamp_in_change_log = 0
```

When deleting an item it is possible to move any references to or from that item to another. Alternatively references can be deleted with the item. The system-wide default behavior may be set here and will be used when the user has not set a preference. Valid options are "remove" or "transfer" Note that you can set per-table defaults by prefacing "delete_reference_handling_default" with a table name. (For example, "ca_objects_delete_reference_handling_default")

```
delete_reference_handling_default = remove
```

### Components

```
ca_objects_container_types = []
ca_objects_component_types = []
ca_objects_component_display_settings = <l>^ca_objects.preferred_labels.name</l> (^ca_
↪objects.idno)
```

### Media

### Media processing tweaks

If you have the PECL Imagick extension installed on your server and don't want to use it with CollectiveAccess (it has a bad habit of choking and crashing on some types of files) you can force CA to ignore it by setting 'dont_use_imagick' to 1; leave it set to zero if you want to use Imagick. When Imagick works, it performs well so you should give it a try and see how it works before disabling support for it.

```
dont_use_imagick = 0
```

If you have ImageMagick or GraphicsMagick installed and PDFs are being inexplicably rejected try setting the corresponding option to 1. It has been observed that ImageMagick chokes on some PDFs. Setting this option will force CA to use Zend_PDF to identify uploaded PDF's, which often resolves the issues at the expense of greater memory consumption.

```
dont_use_imagemagick_to_identify_pdfs = 0
dont_use_graphicsmagick_to_identify_pdfs = 0
```

If you're mostly dealing with large video files or images and don't care about PDF support (or you're using Graphics/ImageMagick for identifying PDFs), you can disable Zend PDF support here. Zend PDF always tries to load the whole fine into memory, which for video files can be several GB and usually results in memory_limit errors.

```
dont_use_zendpdf_to_identify_pdfs = 1
```

CollectiveAccess supports three methods for generating PDF output for download and printing: dompdf (slower; built-in), wkhtmltopdf (faster; requires additional software installation) and phantomjs (faster; requires additional software installation). By default it will favor using wkhtmltopdf if available, falling back to phantomjs and then to dompdf which is always available.

You can override the build in preference and force the use of a specific PDF generator by uncommenting and setting this option to one of the following:

- wkhtmltopdf
- phantomjs
- dompdf

```
use_pdf_renderer = wkhtmltopdf
```

Only media than can be identified by a plugin may be uploaded. If you want to be able to upload any file and have it treated as media, even if the internals of the file cannot be parsed set this to a non-zero value. When set the BinaryFile media plugin is enabled, which will store any unidentifiable uploaded file as binary data. No previews or in-browser viewing will be possible for these files.

```
accept_all_files_as_media = 0
```

PHPs builtin function exif_read_data (http://php.net/manual/en/function.exif-read-data.php) is known to cause unexpected crashes with some files in some versions of PHP, particularly those shipped with RedHat or CentOS Linux. If you experience any weird behavior while processing large files with extensive EXIF metadata, try enabling this setting. If enabled, CollectiveAccess tries to extract metadata using alternate sources like exiftool or GraphicsMagick.

```
dont_use_exif_read_data = 0
```

Alternatively if you experiencing out-of-memory issues while importing media it may well be due to very large EXIF metadata blocked embedded in the file. You can limit the size of metadata to be imported here by specifying the threshold in bytes (Eg. 1048576 = 1mb)

```
dont_use_exif_read_data_if_larger_than = 2097152
```

Files with large embedded metadata blocks may cause out-of-memory errors and/or complicate backup of the datase. You can limit the size of embedded metadata to be extracted during media loading here by specifying the threshold in bytes (Eg. 1048576 = 1mb) Extraction of embedded metadata for media with metadata exceeding the threshold will be skipped. Set to zero or omit if you want all metadata regardless of length to be extracted.

```
dont_extract_embedded_media_metdata_when_length_exceeds = 2097152
```

If you wish to allow the importing of object representation media and icons via http, https and ftp urls set this to 1. Letting users employ your CA installation as a proxy for downloading arbitrary URLs could be seen as a security hole in some cases, so enable this option only if you really need it.

```
allow_fetching_of_media_from_remote_urls = 0
```

If you wish to allow the linking to existing object representations in the manner other relationships set the relevant directives below to 1. Using representations as records that can be targets of relationships can be confusing and, well, odd for many common setups. Still, when you need this behavior you need it, so here it is :-)

```
ca_objects_allow_relationships_to_existing_representations = 0
ca_object_lots_allow_relationships_to_existing_representations = 0
ca_entities_allow_relationships_to_existing_representations = 0
ca_places_allow_relationships_to_existing_representations = 0
ca_occurrences_allow_relationships_to_existing_representations = 0
ca_collections_allow_relationships_to_existing_representations = 0
ca_storage_locations_allow_relationships_to_existing_representations = 0
ca_list_items_allow_relationships_to_existing_representations = 0
ca_loans_allow_relationships_to_existing_representations = 0
ca_movements_allow_relationships_to_existing_representations = 0
```

### Embedded metadata extraction

CA can extract and import metadata embedded in upload media using external applications such as MediaInfo and ExifTool and installation-specific data import mappings. The following options control user interaction and logging for media embedded metadata import.

Users can select the import mapping they wish to use at the time of upload in the editing and batch media importer interfaces when `allow_user_selection_of_embedded_metadata_extraction_mapping` is set to a non-zero value.

When allowing user selection of mappings, `allow_user_embedded_metadata_extraction_mapping_null_option` can be set to include a "no import" option. Setting this option to zero effectively forces import of embedded metadata in all cases.

If it often desirable to have CA automatically select import mappings based upon the format of the uploaded file. The `embedded_metadata_extraction_mapping_defaults` setting can be used to map media file MIME types to mappings. MIME types may be specific (Ex. image/tiff for TIFF format images) or cover entire classes using wildcards (Ex. image/* for images of any type).

```
embedded_metadata_extraction_mapping_defaults = {
    video/* = example_mediainfo_mapping,
    image/* = example_exif_tool_mapping,
    application/pdf = pdf_metadata_import
}
```

The values are the right side of the map must be valid data import mapping codes, as defined in the `code` setting of a mapping worksheet.

How much information is logged when performing an embedded metadata import can be controlled using the `embedded_metadata_extraction_mapping_log_level` setting. Valid values are DEBUG, NOTICE, INFO, WARN, ERR, CRIT and ALERT, where DEBUG logs the most (sometimes too much) information, and levels beyond ERR log only the most critical errors. It is generally best to leave this setting on DEBUG when testing and use NOTICE or INFO if DEBUG is providing too much information.

### Video preview frame generation

You can have CA generate preview frames from uploaded video These settings control how (and if) the preview frames are generated

Should we generate frames? (Set to 1 for yes, 0 for no)

```
video_preview_generate_frames = 1
```

The minimum number of preview frames to generate in any situation CA will adjust timing parameters to ensure at least this number of frames is generated.

```
video_preview_min_number_of_frames = 10
```

The maximum number of preview frames to generate in any situation CA will always stop generating frames when it hits this limit

```
video_preview_max_number_of_frames = 100
```

The time between extracted frames; you can enter this is timecode notation (eg. 10s = 10 seconds; 1:10 = 1 minute, 10 seconds)

```
video_preview_interval_between_frames = 30s
```

The time relative to the start of the video at which to start extracting preview frames; this can be used to ensure you don't generate frames from blank leader footage

```
video_preview_start_at = 2s
```

The time interval relative to the end of the video at which to stop extracting preview frames; this can be used to ensure you don't generate frames from blank footage at the end of a video

```
video_preview_end_at = 2s
```

The time relative to the start of the video at which the "main" video poster preview is being extracted. Express as an absolute time (Ex. 1h 5m 3s) or as a precentage of duration (Ex. 50%)

```
video_poster_frame_grab_at = 5s
```

### Document preview page generation

You can have CA generate preview page images from uploaded documents (only PDFs currently) These settings control how (and if) the preview pages are generated

Should we generate pages? (Set to 1 for yes, 0 for no)

```
document_preview_generate_pages = 1
```

The maximum number of preview pages to generate in any situation CA will always stop generating page images when it hits this limit

```
document_preview_max_number_of_pages = 500
```

The number of pages between extracted pages; set to 1 if you want to generate all pages; set to 10 if you only want to generate every 10th page

```
document_preview_interval_between_pages = 1
```

The page number at which to start extracting pages

```
document_preview_start_page = 1
```

Resolution to rasterize PDF pages with, in DPI

```
document_preview_resolution = 300
```

JPEG quality to rasterize PDF pages with (0-100)

```
document_preview_quality = 95
```

Set to non-zero value if you do not wish to generate representation annotation previews These previews are discrete audio/video files covering a given annotation.

```
dont_generate_annotation_previews = 1
```

### Batch media processing

Root directory of staging area for media import – any media in this directory will appear in media importer file listings

```
batch_media_import_root_directory = <ca_base_dir>/import
```

Allow data importer to pull media from arbitrary directories using paths in the data to be imported. If you don't trust the data being uploaded (or the people doing the uploading) leave this set to zero.

```
allow_import_of_media_from_any_directory = 0
```

```
mediaFilenameToObjectIdnoRegexes = {
    filename_exactly = {
        displayName = _(Filename exactly),
        regexes = { "^(.*)$" }
    },
    filename_without_extension = {
        displayName = _(Filename without extension),
        regexes = { "(.*?)\\.[A-Za-z0-9]+$" }
    },
    filename_with_page_number_included = {
        displayName = _(Filename with page number - page number included),
        regexes = { "(.*?\\.[A-Za-z0-9\\-]+)\\.[A-Za-z]+$", "(.*?)\\.[A-Za-z0-9]+$" }
    },
    filename_with_page_number = {
        displayName = _(Filename with page number - page number stripped),
        regexes = { "(.*?)\\.[A-Za-z0-9\\-]+\\.[A-Za-z]+$" }
    }
}
```

Uncomment and customize the following if you want to transform the names of your media files using Perl-compatible regular expressions (http://pcre.org). The setting is basically a wrapper around PHP's preg_replace function (http://php.net/manual/en/function.preg-replace.php). Each replacement consists of a key (basically a name), a list of "search" regular expressions (usually 1) and a list of "replace" patterns. Both lists must have the same length, i.e. there must be a "replace" pattern for each search regular expression. For more information on the syntax, please refer to the documentation for preg_replace. Note that the media importer will try to mach the results of these replacements to CollectiveAccess records using the "mediaFilenameToObjectIdnoRegexes" list above for each file or directory name IN ADDITION to whatever the original name was. The original file name is matched first.

```
mediaFilenameReplacements = {
    replace_period_w_dash = {
        search = { "([A-Za-z0-9]+)\\.([0-9]+)\\.([A-Za-z0-9]+)" },
        replace = { "$1-$2.$3" }
    },
}
```

List of fields to attempt to match filename-extracted data on Matching will be performed on fields in order, with the first matching record used for import.

You can specify intrinsic field names (Eg. idno), metadata element codes or "preferred_labels" and "nonpreferred_labels" to match on labels

```
batch_media_import_match_on = [idno]
```

### Batch metadata import

```
batch_metadata_import_log_directory = <ca_base_dir>/app/log
```

Directory to temporarily stash ajax-based uploads of media in

```
ajax_media_upload_tmp_directory = <ca_app_dir>/tmp
```

Max time in seconds to let media live in tmp directory before it can be removed

```
ajax_media_upload_tmp_directory_timeout = 86400
```

### Object representation download options

Media versions to provide downloads of

```
ca_object_representation_download_versions = [original, large, medium, small]
```

Set maximum number of files to allow to be downloaded in one go. Leave set to 0 or blank for no limit. maximum_download_file_count =

### Task queue set up (deferred processing of uploaded media)

```
taskqueue_handler_plugins = <ca_lib_dir>/Plugins/TaskQueueHandlers
taskqueue_tmp_directory = <ca_app_dir>/tmp
taskqueue_max_opo_processes = 4
taskqueue_process_timeout = 3600
taskqueue_max_items_processed_per_session = 100
```

### Admin

Nit picky stuff related to system configuration and administration.

### Character set to use (usually utf-8; might be ISO-8859-1)

```
character_set = utf-8
```

### System configuration check options (under "Manage" > "Administrate" > "Configuration Check")

> The configuration check can do a thorough, but time consuming, check of file permissions and other settings. These checks can be useful but on some servers, especially those using file systems mounted over a network, they can be very slow. If you are on such a server you can disable all "expensive" configuration checks here.

```
dont_do_expensive_configuration_checks_in_web_ui = 0
```

### Configuration exporter options

```
configuration_export_only_system_displays = 1
configuration_export_only_system_search_forms = 1
```

Exclude lists from configuration export with more than a specified number of items. If set to zero no limit is enforced.

```
configuration_export_exclude_lists_larger_than = 0
```

list of list codes to exclude from configuration export

```
configuration_export_exclude_lists = []
```

### Object lot inheritance

don't inherit lot relationship from parent object

```
ca_objects_dont_inherit_lot_id_from_parent = 0
```

### Restrict editing of codes for list and metadata elements

Allowing free editing and code and data type settings can result in invalid configuration. The ability to edit these values once set can be restricted here.

```
ca_lists_dont_allow_editing_of_codes_when_in_use = 0
ca_list_items_dont_allow_editing_of_codes_when_in_use = 0
ca_metadata_elements_dont_allow_editing_of_codes_when_in_use = 0
ca_metadata_elements_dont_allow_editing_of_data_types_when_in_use = 0
```

### SMS notifications

```
enable_sms_notifications = 0
```

Each SMS plugin supports a specific gateway. For now only SendHub.com is supported.

```
sms_plugin = SendHub
sms_user = MY_SENDHUB_USERNAME
sms_api_key = MY_SENDHUB_API_KEY
```

### Session settings

```
session_lifetime = 31536000
session_domain =
```

### Email notifications

Settings for notifications system used for metadata-based alerts

```
notification_email_sender = no-reply@<site_hostname>
notification_email_subject = (<app_display_name>) Metadata Notification from␣
↪CollectiveAccess
```

### Export

### File names for data export download files

If the given display template doesn't yield a usable result, the exporter falls back to relatively nondescript defaults single item exports via inspector in the corresponding editor

```
ca_objects_single_item_export_filename = ^ca_objects.idno
ca_object_lots_single_item_export_filename = ^ca_object_lots.idno_stub
ca_entities_single_item_export_filename = ^ca_entities.idno
ca_places_single_item_export_filename = ^ca_places.idno
ca_occurrences_single_item_export_filename = ^ca_occurrences.idno
ca_collections_single_item_export_filename = ^ca_collections.idno
ca_lists_single_item_export_filename = ^ca_lists.list_code
ca_list_items_single_item_export_filename = ^ca_list_items.idno
ca_loans_single_item_export_filename = ^ca_loans.idno
ca_movements_single_item_export_filename = ^ca_movements.idno
ca_object_representations_single_item_export_filename = ^ca_object_representations.
↪idno
ca_representation_annotations_single_item_export_filename = ^ca_representation.
↪annotations.annotation_id
ca_storage_locations_single_item_export_filename = ^ca_storage_locations.idno
ca_tours_single_item_export_filename = ^ca_tours.tour_code
ca_tour_stops_single_item_export_filename = ^ca_tours_stops.idno
```

batch exports via sets or browse results

```
ca_objects_batch_export_filename = objects_batch_export
ca_object_lots_batch_export_filename = lots_batch_export
ca_entities_batch_export_filename = entities_batch_export
ca_places_batch_export_filename = places_batch_export
ca_occurrences_batch_export_filename = occurrences_batch_export
ca_collections_batch_export_filename = collections_batch_export
ca_lists_batch_export_filename = lists_batch_export
```

(continues on next page)

(continued from previous page)

```
ca_list_items_batch_export_filename = list_items_batch_export
ca_loans_batch_export_filename = loans_batch_export
ca_movements_batch_export_filename = movements_batch_export
ca_object_representations_batch_export_filename = representations_batch_export
ca_representation_annotations_batch_export_filename = annotations_batch_export
ca_storage_locations_batch_export_filename = storage_locations_batch_export
ca_tours_batch_export_filename = tours_batch_export
ca_tour_stops_batch_export_filename = tour_stops_batch_export
```

List of alternate destinations for data exports. The only supported type for now is ‘github’.

For GitHub repositories it’s highly recommended to *not* enter your main account password here but to use a personal access token instead. You can create it in the GitHub account settings under “Applications”>”Personal Access Tokens”. The token has to have ‘repo’ access.

```
exporter_alternate_destinations = {
    github = {
        type = github,
        display = GitHub repository,
        user credentials
        username = your_github_username,
        token = enter_access_token_here,
        repository information
        owner = enter_repository_owner,
        repository = collectiveaccess_export,
        base_dir = exports/from_ca,
        branch = master,
        update_existing = 1
    },
}
```

### You’re done…

….probably. Most users don’t modify the configs below.

### URL configuration (paths to controllers and themes)

```
auth_login_path = system/auth/login
auth_login_url = <ca_url_root>/index.php/system/auth/login
auth_logout_url = <ca_url_root>/index.php
controllers_directory = <ca_app_dir>/controllers
```

Url path to error display page; user will be directed here upon unrecoverable error (eg. bad controller or action)

```
error_display_url = <ca_url_root>/index.php/system/Error/Show
```

Url to redirect user to when nothing is specified (eg. they go to /index.php) ONLY PUT THE CONTROLLER/ACTION PATH HERE - leave out the ‘index.php’

```
default_action = /Dashboard/Index
```

Services

```
service_controllers_directory = <ca_app_dir>/service/controllers
service_default_action = /search/rest/doSearch
service_view_path = <ca_app_dir>/service/views
```

### Paths to other config files

```
data_model = <ca_conf_dir>/datamodel.conf
user_pref_defs = <ca_conf_dir>/user_pref_defs.conf
external_applications = <ca_conf_dir>/external_applications.conf
media_volumes = <ca_conf_dir>/media_volumes.conf
file_volumes = <ca_conf_dir>/file_volumes.conf
default_media_icons = <ca_conf_dir>/default_media_icons.conf
search_config = <ca_conf_dir>/search.conf
browse_config = <ca_conf_dir>/browse.conf
media_processing_settings = <ca_conf_dir>/media_processing.conf
annotation_type_config = <ca_conf_dir>/annotation_types.conf
attribute_type_config = <ca_conf_dir>/attribute_types.conf
application_monitor_config = <ca_conf_dir>/monitor.conf
assets_config = <ca_conf_dir>/assets.conf
bundle_type_config = <ca_conf_dir>/bundle_types.conf
xml_config = <ca_conf_dir>/xml.conf
user_actions = <ca_conf_dir>/user_actions.conf
find_navigation = <ca_conf_dir>/find_navigation.conf
media_display = <ca_conf_dir>/media_display.conf
media_metadata = <ca_conf_dir>/media_metadata.conf
access_restrictions = <ca_conf_dir>/access_restrictions.conf
datetime_config = <ca_conf_dir>/datetime.conf
authentication_config = <ca_conf_dir>/authentication.conf
services_config = <ca_conf_dir>/services.conf
visualization_config = <ca_conf_dir>/visualization.conf
prepopulate_config = <ca_conf_dir>/prepopulate.conf
linked_data_config = <ca_conf_dir>/linked_data.conf
```

Path to navigation config file - defines menu structure

```
nav_config = <ca_conf_dir>/navigation.conf
```

OAI configuration

```
oai_harvester_config = <ca_conf_dir>/oai_harvester.conf
oai_provider_config = <ca_conf_dir>/oai_provider.conf
```

### Path to application plugins

```
application_plugins = <ca_app_dir>/plugins
```

### Path to dashboard widgets

```
dashboard_widgets = <ca_app_dir>/widgets
```

### Password reset parameters

```
password_reset_url = <site_host><ca_url_root>/index.php?action=reset_password&form_
↪action=reset
```

### ID numbering (for objects, object lots and authorities)

```
multipart_id_numbering_config = <ca_conf_dir>/multipart_id_numbering.conf
```

### Media and file processing paths

```
media_plugins = <ca_lib_dir>/Plugins/Media
file_plugins = <ca_lib_dir>/Plugins/File
```

Directory to use for Tilepic generation temporary files

```
tilepic_tmpdir = <ca_app_dir>/tmp
```

Name of plugin class to use for id number field in objects, object lots and authorities that support id numbering (entities, places, collections and occurrences)

```
ca_objects_id_numbering_plugin = MultipartIDNumber
ca_object_lots_id_numbering_plugin = MultipartIDNumber
ca_entities_id_numbering_plugin = MultipartIDNumber
ca_places_id_numbering_plugin = MultipartIDNumber
ca_collections_id_numbering_plugin = MultipartIDNumber
ca_occurrences_id_numbering_plugin = MultipartIDNumber
ca_list_items_id_numbering_plugin = MultipartIDNumber
ca_loans_id_numbering_plugin = MultipartIDNumber
ca_movements_id_numbering_plugin = MultipartIDNumber
ca_tours_id_numbering_plugin = MultipartIDNumber
ca_tour_stops_id_numbering_plugin = MultipartIDNumber
ca_object_representations_id_numbering_plugin = MultipartIDNumber
ca_storage_locations_id_numbering_plugin = MultipartIDNumber
ca_site_pages_id_numbering_plugin = MultipartIDNumber
ca_site_page_media_id_numbering_plugin = MultipartIDNumber
```

### Formats for form elements

If set text of "required_field_marker" will be displayed for bundles in editors for which input is required

```
show_required_field_marker = 0
```

Text to display for bundles in editors for which input is required

> required_field_marker = <span style="color: bb0000; font-size:10px; font-weight: bold;">(Required)
> </span>

These are used to format data entry elements in various editing formats. Don't change them unless you know what you're doing Used for intrinsic fields (simple fields)

```
form_element_display_format = <div class='formLabel'>^EXTRA^LABEL<br/>^ELEMENT</div>
form_element_display_format_without_label = <div class='formLabel'>^ELEMENT</div>
form_element_error_display_format = <div class='formLabel'>^EXTRA^LABEL (<span class=
↪'formLabelError'>^ERRORS</span>)<br/>^ELEMENT</div>
```

Used for bundle-able fields such as attributes

```
bundle_element_display_format = <div class='bundleLabel'>^LABEL ^DOCUMENTATIONLINK ^
↪ELEMENT</div>
bundle_element_display_format_without_label = <div class='formLabel'>^ELEMENT</div>
bundle_element_error_display_format = <div class='bundleLabel'>^LABEL (<span class=
↪'bundleLabelError'>^ERRORS</span>)<br/>^ELEMENT</div>
```

Used for the 'idno' field of bundle-providers (Eg. ca_objects, ca_places, etc.)

```
idno_element_display_format = <div class='formLabel'>^LABEL<br/>^ELEMENT <span id=
↪'idnoStatus'></span></div>
idno_element_display_format_without_label = <div class='formLabel'>^ELEMENT <span id=
↪'idnoStatus'></span></div>
idno_element_error_display_format = <div class='formLabel'>^LABEL (<span class=
↪'formLabelError'>^ERRORS</span>)<br/>^ELEMENT <span id='idnoStatus'></span></div>
```

#### Proxy server configuration for web services

In some larger networks servers are required to run their HTTP/HTTPS requests through a proxy server. If this applies to your setup, uncomment the following lines and enter your proxy configuration here.

```
web_services_proxy_url = tcp://127.0.0.1:8080
```

### 1.18.2 Browse.conf

CollectiveAccess includes a configurable *Browse Engine* that powers all faceted browse and search features. The engine is capable of browsing for, and returning sets of, any of the primary item types: objects, object lots, entities, places, occurrences, collections and storage locations. The engine automatically caches both results and generated facet content to improve performance. If two users perform the same browse, results for the second browse will be picked up from the cache saving time. Similarly, facet content, which is often costly to generate, is shared across similar browses increasing responsiveness.

#### Current use

Faceted browse is currently used in the Providence (back-end) "Find" interfaces for all primary item types. It is also used to provide browse services in the Pawtucket public-access front-end.

#### Configuration

Any intrinsic field (ie. a field that is always part of an item such as extent and extent_units in object lots), metadata attribute or related authority may be used for browsing. Since every deployment of CA is different, and the metadata schema varies from one installation to another, you must tell the browse engine what sorts of information you want to be browse-able and how that data should be displayed to the user. This is done by modifying the browse configuration file in *app/conf/browse.conf*. As with all other CA configuration files, *browse.conf* is written using the standard CA configuration syntax.

| Top level key | Description |
|---|---|
| cache_timeout | Number of seconds to keep information for a cached browse around before discarding. The suggested value for this key depends upon how the browse is being uses. A cached browse will not reflect changes made to the data since its creation, so for a busy backend system this value should be relative low: 300 seconds (5 minutes) is a reasonable value. For front-end systems where the catalogue data is not changing often, 86400 seconds (1 day) may be a more appropriate value. If you want to disable caching set this key to 0. |
| <browse_table_name> | For each item type you wish to support browsing on you must define an associative array attached to the item table name: ca_objects, ca_entities, ca_places, ca_occurrences, ca_collections, ca_object_lots, and ca_storage_locations. The values set in each item array define the behavior of the browse for that item type. These values are defined below. |

For each item type you want to be browse-able, you must define a top-level key with the item's table name (eg. ca_objects for objects) and as associative array value. The array must contain a facets key whose value is in turn an associative array defining each available browse facet for the item type. The keys of the facets array are arbitrary code name for the facets, it doesn't matter what they are so long as they are unique within the facet list. The values are yet another associative array which actually defines the characteristics of the facet.

Each facet has a type and some (but not all) of the facet definition keys are dependent upon this type. The follow types of facets are currently supported:

| Facet type | Description |
|---|---|
| authority | Authority facets allow for browsing on cataloguing applied to the browsed item from a related authority. Eg. if you want to browse for objects by place name, you'd set up a facet of type authority with options to cover the places authority. |
| field | Allows browsing on items using an intrinsic field. These include idno (identifier) and the handful legacy intrinsics such as ca_objects extent. |
| fieldList | Allows browsing on lists that are directly related to an item via an intrinsic field. These include the type lists for each item type (eg. browse by object types), access and workflow status. |
| normalizedDates | Allows browsing on date attributes where the values have been normalized, or adjusted to span periods of time. Since dates are often specific to the day (or even hour, minute and second), browsing on unmodified date data is usually undesirable. normalizedDate facets will return browse choices where dates have been collapsed into days, months, years, decades or centuries. |
| attribute | Allows browsing on any simple single-value attribute. Values are presented as-is, so you should only configure browsing on metadata elements that have a relatively small range of possible values. Browsing on an attribute with narrative text content will not work well. Browsing on an attribute with typed-in text indicating materials or location may work well if data quality is relatively high. |
| label | Allows browsing on preferred and non-preferred labels associated with the browsed item. This can be useful for creating index-like lists of titles for a given type of item. Since the facet will list all unique label values for a type of item it is mainly useful in smaller systems with relatively few items or where records share a manageable number of labels. For larger systems with many distinct label values this facet type is likely to provide poor usability and performance. |
| has | Provides a means to browse for items that either have at least one relationship to some other type of item, or for items with no relationships to a specified type of item. The option values for this facet are always "yes" and "no." This type of facet is primarily useful for retrieving objects with or without object representations (media), but it can also be used for reporting and data quality assessment purposes. For example, this facet could be configured to all retrieval of all objects without an associated entity. |
| dupeidno | Allows browsing for records with identifiers that are duplicated by at least one other record. Facet values are the number of repeats an identifier has. This type of facet is useful for facilitating clean up of data where identifiers have been erroneously reused. Available from version 1.7.7 |
| location | Allows browsing for records on current location, as defined in the current location policy configuration (current_location_criteria in app.conf). |
| violations | Allows browsing for records that are in violation of rules in the metadata dictionary. (Note that the metadata dictionary is currently an undocumented feature with no configuration UI). |
| checkouts | Allows browsing for objects based upon checkout status set by the library check in/out module. |
| inHomeLocation | Allows browsing for objects based upon whether they are currently in their home location or not, as defined in the current location policy configuration (current_location_criteria in app.conf). (Available from version 1.7.9) |

For all types of facets the following configuration keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| indefinite_article | The indefinite article to use when displaying the facet label. | Yes |
| multiple | If this directive is set to 1 the facet will perform an OR browse instead of an AND browse. With “multiple” a user is able to select more than one value per facet, for example, in a date browse a user can find records cataloged as 1990 or 1991. | No |
| label_singular | The name to display for this facet, in the singular (eg. “place name”). | Yes |
| label_plural | The name to display for this facet, in the plural (eg. “place names”). | Yes |
| description | Description of the facet. This text appears on browse landing page in Pawtucket | No |
| group_mode | The method by which to group facet values for display. Currently ‘alphabetical’, ‘hierarchical’ and ‘none’ groupings are supported. ‘Alphabetical’ mode groups items alphabetically by the first letter in their name; ‘hierarchical’ displays hierarchical facets (authorities only) in an interactive hierarchy browser; ‘none’ simply lists items out. | No |
| individual_group_display | If set to a non-zero value, each group in the facet will be displayed separately and loaded on-demand. This can improve performance in some cases. | No |
| type_restrictions | An optional list of browse item types to restrict use of this facet to. This is useful if you are restricting your browses to specific types of browse items (ex. occurrences of type “exhibitions” only) and want to have different facet configurations for each type-specific browse. By restricting a facet to “exhibitions”, for example, you ensure that it will only appear when you are browsing specifically for exhibitions, or if you are performing an unrestricted browse. You can specify the restrictions are numeric type_id’s or alphanumeric type codes. The latter is generally preferred as it is easier to read, understand and maintain. | No |
| requires | An optional list of facet names, at least one of which must appear in the browse criteria (ie. have been used in the current browse), for the facet to be available. This is useful if you have a facet that is only relevant in the context of a selection limited by another facet. For example, if you have a facet defined for countries and another for state/provinces, you can ensure that the state/province facet is only made available to the user if a country has already been selected for the browse by making state/providence “require” the country facet. | No |
| single_value | If set to a valid facet value, then selecting the facet will browse on the specified value rather than opening up a browse facet display with choices. This option can be useful for facets with one or two values, where explicit links are preferred over a list interface. | No |
| facet_groups | A list of tags (“groups”) to apply to the facet. Facets displayed by the browse engine can be limited using these tags, via the browse engine setFacetGroup() method. Facet groups are mainly useful when you need to have distinct sets of facets display in different contexts. | No |
| relative_to | Browse facets are typically generated relative to the table being browsed. If you are browsing objects, for example, a facet of related entities will display names of entities related directly to objects. relative_to, when set to a table other than the table being browsed, enables you to generate facets with values from a table related to the one you are browsing. You could, for example, generate a facet of birth dates of entities related to objects, or a list of places related to entities related to objects. In effect this is an indirect browse. Even though the facet values are related to the relative_to table, the results of the browse are still the same. relative_to can be very useful when metadata you want to browse by is actually related to a table different than the one you wish to get results for. Value must be a valid item table name (Eg. ca_objects, ca_entities, ca_places, ca_occurrences, ca_collections, ca_storage_locations, ca_list_item, Etc.) | No |
| access | A list of access values to restrict facet visibility to. Access values are numerical values used to control public visibility of records. Each user, authenticated or not, while be associated with one or more access values. For non-authenticated users access values will be those specified in the public_access_settings directive in app.conf. For authenticated users it will be a list of values conferred by the users associated roles. Any user having any of the access values for the facet will be able to use the facet. If this option is omitted or left empty no restriction will be applied. Available from | No |

For facets of type **authority** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| table | The authority table to browse by (Eg. ca_objects, ca_entities, ca_places, ca_occurrences, ca_collections, ca_storage_locations, ca_list_item) | Yes |
| relationship_table | The "linking table" between the item type your are browsing for and the authority you are browsing by. For example, if you are browsing for objects by entities this table is ca_objects_x_entities. See the installation profile manual for a full list of these table names. | Yes |
| restrict_to_types | An optional list of types *for the item you are browsing by* to restrict the facet to. The types can be internal item_ids for the types or ca_list_item.idno values (eg. list item codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places of type 'river' for instance. | No |
| restrict_to_relationship_types | An optional list of relationship types to restrict the facet to. The types can be internal relationship type_ids for the relationship types or alphanumeric type codes (eg. type codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places linked to objects with relation type 'depicts') | No |
| exclude_relationship_types | An optional list of relationship types to exclude from the facet. The types can be internal relationship type_ids for the relationship types or alphanumeric type codes (eg. type codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places linked to objects with relation types other than 'depicts') | No |
| generate_facets_for_types | If set to a non-zero value, will cause the current facet to be automatically converted into a separate facet for each type of the item type being browsed by. This option is typically employed to provide browsing of occurrences where the various types are unrelated, but you can also use this on other authorities to provide a fine-grained browse without having to hardcode the type hierarchy into the configuration. | No |
| show_all_when_first_facet | If set to non-zero value, will force this facet to include all items in the authority whether than are related to the underlying browse table or not when the browse has no criteria set (ie. when the facet is the first one chosen in the browse). Default is false. | No |
| order_by_label_fields | A list of fields in authority label table to sort by. You can do multi-level sorting by specifying more than one field in the list. Ascending sort order is assumed. The list should be only field names; do not include the table name | No |
| group_mode | The method by which to group facet values for display. Currently 'alphabetical', 'hierarchical' and 'none' groupings are supported. 'Alphabetical' mode groups items alphabetically by the first letter in their name; 'hierarchical' displays hierarchical facets (authorities only) in an interactive hierarchy browser; 'none' simply lists items out. | No |
| show_hierarchy | Set to non-zero value to display hierarchy on items in this facet; default is to not display hierarchy | No |
| hierarchical_delimiter | Character(s) to place between elements of the hierarchy | No |
| remove_first_items | Number of items to trim off the top of the hierarchy (leave blank or set to 0 to trim nothing) | No |
| hierarchy_limit | Maximum length of hierarchy to display (leave blank to return hierarchy unabridged) | No |
| hierarchy_order | Direction to display hierarchy in. Can be ASC or DESC (default is DESC). ASC displays the root first; DESC displays the lowest element in the hierarchy first | No |

For facets of type **field** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| field | The field in the item type being browsed to browse by (Eg. idno, extent) | Yes |

For facets of type **fieldList** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| field | The field in the item type being browsed to browse by (Eg. type_id, item_status_id, access, status) | Yes |
| display | The field to display in the facet. If not set the preferred label is displayed. | No |

For facets of type **normalizedDates** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| element_code | The element code of the metadata element to be browsed. Must have an attribute type of DateRange | Yes |
| normalization | Sets the method used to normalize date values. Supported values are days, months, years, decades, centuries. | Yes |
| sort | Sets the order in which to sort the returned dates. A value of 'DESC' will sort the dates in descending order (most recent first), which a value of 'ASC' will sort in ascending order (oldest first). The default if this is not specified is ascending order. | No |
| minimum_date | If set, the facet will only include dates that occur after the specified value. The value can be any valid date expression (eg. 1900, 12/7/1914, etc.). | No |
| maximum_date | If set, the facet will only include dates that occur before the specified value. The value can be any valid date expression (eg. 1900, 12/7/1914, etc.). | |
| treat_before_dates_as_circa | If set facet will treat after "before xxxx" dates as circa dates. Available from version 1.7.7 | No |
| treat_after_dates_as_circa | If set facet will treat after "after xxxx" dates as circa dates. Available from version 1.7.7 | No |
| include_unknown | If set facet will include an "unknown date" option to find records without any associated date. Available from version 1.7.8 | |

For facets of type **attribute** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| element_code | The element code of the metadata element to be browsed. Only attributes of type List and Text are officially supported, but Numeric and Currency types tend to work well too. Other types may or may not work as you would like (but usually will). For container elements, use the sub-element's element code. | Yes |
| suppress | A list of values to omit from the facet. If the metadata element is a list these should be list item_id's or idno's. For other element types use the label value. | No |

For facets of type **label** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| restrict_to_types | An optional list of types *for the item you are browsing by* to restrict the facet to. The types can be internal item_ids for the types or ca_list_item.idno values (eg. list item codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places of type 'river' for instance. | No |
| preferred_labels_only | If set to a non-zero value, will force this facet to include only preferred labels. Default is false: include all labels, preferred and non-preferred. | No |
| group_mode | The method by which to group facet values for display. Currently only 'alphabetical' and 'none' groupings are supported. This mode groups items alphabetically by the first letter in their name. | No |
| order_by_label_fields | Available from version 1.7.6. A list of fields in the label table to sort by. You can do multi-level sorting by specifying more than more field in the list. Ascending sort order is assumed. The list should be only field names; do not include the table name | No |
| template | Available from version 1.7.6. A template to format returned labels with. This is primarily useful for formatting entity labels, which are comprised of several component fields (surname, forename, middle name, displayname, Etc.). The template is text with label components specified by the field name of the component preceded by a caret ("^"). For example, to format an entity label surname-comma-forename style the template is "^surname, ^forename". If no template is specified the primary display field for the label is used. | No |

For facets of type **has** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| table | The authority table to look for relationships to (Eg. ca_objects, ca_entities, ca_places, ca_occurrences, ca_collections, ca_storage_locations, ca_list_item) | Yes |
| relationship_table | The "linking table" between the item type your are looking for relationships to and the authority you are browsing by. For example, if you are looking for objects with object representations this table is ca_objects_x_object_representations. | Yes |
| restrict_to_types | An optional list of types *for the item you are browsing by* to restrict the facet to. The types can be internal item_ids for the types or ca_list_item.idno values (eg. list item codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places of type 'river' for instance. | No |
| restrict_to_relationship_types | An optional list of relationship types to restrict the facet to. The types can be internal relationship type_ids for the relationship types or alphanumeric type codes (eg. type codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places linked to objects with relation type 'depicts'. | No |
| exclude_relationship_types | An optional list of relationship types to exclude from the facet. The types can be internal relationship type_ids for the relationship types or alphanumeric type codes (eg. type codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places linked to objects with relation types other than 'depicts'. | No |
| label_yes | Text to use for "yes" value facet option. If no specified defaults to "Yes". | No |
| label_no | Text to use for "no" value facet option. If no specified defaults to "No". | No |

For facets of type **dupeidno** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| restrict_to_types | An optional list of types *for the item you are browsing by* to restrict the facet to. The types can be internal item_ids for the types or ca_list_item.idno values (eg. list item codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places of type 'river' for instance. | No |
| exclude_types | An optional list of types to exclude from the facet. The types can be internal item_ids for the types or ca_list_item.idno values (eg. list item codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: all places with type other than 'river' for instance. | No |

For facets of type **location** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| display | Dictionary of display parameters for individual kinds of locations. The format is similar to that uses in app.conf for the current_location_criteria policy. In general you can set display templates for each kind of location using a format like this:<br>`display = {`<br>`   ca_storage_locations =`<br>`↪{`<br>`      related = {`<br>`         template = ^ca_`<br>`↪storage_locations.`<br>`↪hierarchy.preferred_`<br>`↪labels%delimiter=___`<br>`      }`<br>`   },`<br>`},`<br>where first level keys are table names and second level keys are types. If omitted the configuration set in app.conf in current_location_criteria is used. | No |
| collapse | Dictionary controlling which sorts of locations are collapsed into general headings rather than displayed individually. Keys of the dictionary are table names and type separated with a slash ("/"). Values are text with which to represent the collapsed group in the browse facet. For example, to collapse all occurrences of type "exhibition" into a single facet value labeled "On loan" use:<br>`collapse = {`<br>`  ca_occurrences/`<br>`↪exhibition = On loan`<br>`}`<br>Selecting "On loan" would return all objects where the current location is any exhibition. Without the collapse setting, each exhibition would be listed individually. | No |
| maximumBrowseDepth | The element code of the metadata element to be browsed. Must have an attribute type of DateRange | No |
| policy | The current value tracking policy to use. If not specified the default policy is used. | |

For facets of type **inHomeLocation** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| restrict_to_types | An optional list of types *for the item you are browsing by* to restrict the facet to. The types can be internal item_ids for the types or ca_list_item.idno values (eg. list item codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places of type 'river' for instance. | No |
| exclude_types | An optional list of types to exclude from the facet. The types can be internal item_ids for the types or ca_list_item.idno values (eg. list item codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: all places with type other than 'river' for instance. | No |
| policy | The current value tracking policy to use. If not specified the default policy is used. | |

For facets of type **violations** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| restrict_to_types | An optional list of types for the item you are browsing by to restrict the facet to. The types can be internal item_ids for the types or ca_list_item.idno values (eg. list item codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places of type 'river' for instance. | Yes |

For facets of type **checkouts** these additional keys are defined:

| Facet key | Description | Mandatory? |
|---|---|---|
| restrict_to_types | An optional list of types *for the item you are browsing by* to restrict the facet to. The types can be internal item_ids for the types or ca_list_item.idno values (eg. list item codes set by the installation profile). This key lets you set up facets that only browse a subset of a given authority: only places of type 'river' for instance. | No |
| mode | Determines what checkouts are browsed on. If set to "user" checkouts are shown by user (subject to type determined by the 'status' option). If set to "all" all types of checkouts are shown in facet. In this case 'status' is not used. | Yes |
| status | Limits facet to a specific type of checkout when mode is set to "user". Must be one of "all", "available", "out", "reserved", "overdue". Default is "all" when omitted. | No |

### Browse results when no criteria are defined

By default the browse will not return results if you attempt to execute a browse with no criteria defined. In principle, a criteria-less browse should return all possible results – every item in your database. However, for most data sets such a result set would be of limited use and slow to render. In most CA Providence and Pawtucket implementations, a special "start browsing" display is used when no criteria are defined.

If you really do want all results returned when no criteria are defined you can force it on a per-table basis by setting show_all_for_no_criteria_browse in the table-level block (the one that must contain the facets list). See the ca_objects block in the example below to see how this is done.

### Avoiding Cache Confusion

Browse results are cached for a period of time defined by the cache_timeout value in your browse configuration. Once cached, a browse result will be reused until it expires, even if you change your browse configuration in the meantime. This has the effect of making it almost impossible to experiment with browse configuration while caching is enabled. If you are developing or debugging a browse configuration, be sure to set cache_timeout to zero while you're working. Once your browse is working as you want it to re-enable the cache by setting the timeout to a reasonable value. Caching significantly improves overall performance so you'll probably want it enabled for every day use.

### Example Configuration

A working browse.conf should look something like this:

```
# Browse configuration

# number of seconds to keep cached browses around
# set to 0 to disable caching
cache_timeout = 60

# Configuration for object browse
ca_objects = {
                show_all_for_no_criteria_browse = 1,
        facets = {
                entity_facet = {
                        # 'type' can equal authority, attribute, fieldList,␣
↪normalizedDates
                        type = authority,
                        table = ca_entities,
                        relationship_table = ca_objects_x_entities,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [surname, forname],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(entity),
                        label_plural = _(entities)
                },
                place_facet = {
                        type = authority,
                        table = ca_places,
                        relationship_table = ca_objects_x_places,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(place),
                        label_plural = _(places)
                },
                collection_facet = {
                        type = authority,
                        table = ca_collections,
                        relationship_table = ca_objects_x_collections,
                        restrict_to_types = [],
```

(continues on next page)

(continued from previous page)

```
                restrict_to_relationship_types = [],
                sort_by = [name],
                group_mode = alphabetical,

                indefinite_article = a,
                label_singular = _(collection),
                label_plural = _(collections)
        },
        occurrence_facet = {
                type = authority,
                table = ca_occurrences,
                generate_facets_for_types = 1,
                relationship_table = ca_objects_x_occurrences,
                restrict_to_types = [],
                restrict_to_relationship_types = [],
                sort_by = [name],
                group_mode = alphabetical,

                indefinite_article = an,
                label_singular = _(occurrence),
                label_plural = _(occurrences)
        },
        term_facet = {
                type = authority,
                table = ca_list_items,
                relationship_table = ca_objects_x_vocabulary_terms,
                restrict_to_types = [],
                restrict_to_relationship_types = [],
                sort_by = [name],
                group_mode = alphabetical,

                indefinite_article = a,
                label_singular = _(term),
                label_plural = _(terms)
        },
        type_facet = {
                type = fieldList,
                field = type_id,
                sort_by = [name],
                group_mode = alphabetical,

                indefinite_article = a,
                label_singular = _(type),
                label_plural = _(types)
        },
        object_subtype_facet = {
                type = attribute,
                element_code = object_subtypes,

                requires = type_facet,
                group_mode = alphabetical,

                label_singular = _("Sub-Type"),
                label_plural = _("Sub-Types")
        },
        status_facet = {
                type = fieldList,
```

(continues on next page)

(continued from previous page)

```
                        field = status,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(status),
                        label_plural = _(statuses)
                },
                access_facet = {
                        type = fieldList,
                        field = access,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(access status),
                        label_plural = _(access statuses)
                },
                date_facet = {
                        type = normalizedDates,
                        element_code = creation_date,

                        # 'normalization' can be: years, decades, centuries
                        normalization = years,
                        sort_by = [name],
                        group_mode = none,

                        indefinite_article = a,
                        label_singular = _(year),
                        label_plural = _(years)
                }
        }
}

# Configuration for object lot browse
ca_object_lots = {
        facets = {
                entity_facet = {
                        # 'type' can equal authority, attribute, fieldList,␣
↪normalizedDates
                        type = authority,
                        table = ca_entities,
                        relationship_table = ca_object_lots_x_entities,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [surname, forname],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(entity),
                        label_plural = _(entities)
                },
                place_facet = {
                        type = authority,
                        table = ca_places,
                        relationship_table = ca_object_lots_x_places,
                        restrict_to_types = [],
```

(continues on next page)

(continued from previous page)

```
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(place),
                        label_plural = _(places)
                },
                collection_facet = {
                        type = authority,
                        table = ca_collections,
                        relationship_table = ca_object_lots_x_collections,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(collection),
                        label_plural = _(collections)
                },
                occurrence_facet = {
                        type = authority,
                        table = ca_occurrences,
                        relationship_table = ca_object_lots_x_occurrences,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(occurrence),
                        label_plural = _(occurrences)
                },
                term_facet = {
                        type = authority,
                        table = ca_list_items,
                        relationship_table = ca_object_lots_x_vocabulary_terms,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(term),
                        label_plural = _(terms)
                },
                type_facet = {
                        type = fieldList,
                        field = type_id,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(type),
                        label_plural = _(types)
                },
```

(continues on next page)

(continued from previous page)

```
                status_facet = {
                        type = fieldList,
                        field = status,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(status),
                        label_plural = _(statuses)
                },
                access_facet = {
                        type = fieldList,
                        field = access,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(access status),
                        label_plural = _(access statuses)
                }
        }
}
# ----------------------------------------------------------------------
# Configuration for entity browse
ca_entities = {
        facets = {
                place_facet = {
                        type = authority,
                        table = ca_places,
                        relationship_table = ca_entities_x_places,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(place),
                        label_plural = _(places)
                },
                occurrence_facet = {
                        type = authority,
                        table = ca_occurrences,
                        relationship_table = ca_entities_x_occurrences,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(occurrence),
                        label_plural = _(occurrences)
                },
                collection_facet = {
                        type = authority,
                        table = ca_collections,
                        relationship_table = ca_entities_x_collections,
                        restrict_to_types = [],
```

(continues on next page)

(continued from previous page)

```
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(collection),
                        label_plural = _(collections)
                },
                term_facet = {
                        type = authority,
                        table = ca_list_items,
                        relationship_table = ca_entities_x_vocabulary_terms,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(term),
                        label_plural = _(terms)
                },
                type_facet = {
                        type = fieldList,
                        field = type_id,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(type),
                        label_plural = _(types)
                },
                status_facet = {
                        type = fieldList,
                        field = status,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(status),
                        label_plural = _(statuses)
                },
                access_facet = {
                        type = fieldList,
                        field = access,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(access status),
                        label_plural = _(access statuses)
                }
        }
}
# ----------------------------------------------------------------------
# Configuration for collection browse
ca_collections = {
        facets = {
```

(continues on next page)

(continued from previous page)

```
                entity_facet = {
                        # 'type' can equal authority, attribute, fieldList,␣
↪normalizedDates
                        type = authority,
                        table = ca_entities,
                        relationship_table = ca_entities_x_collections,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [surname, forname],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(entity),
                        label_plural = _(entities)
                },
                place_facet = {
                        type = authority,
                        table = ca_places,
                        relationship_table = ca_places_x_collections,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(place),
                        label_plural = _(places)
                },
                occurrence_facet = {
                        type = authority,
                        table = ca_occurrences,
                        relationship_table = ca_occurrences_x_collections,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(occurrence),
                        label_plural = _(occurrences)
                },
                term_facet = {
                        type = authority,
                        table = ca_list_items,
                        relationship_table = ca_collections_x_vocabulary_terms,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(term),
                        label_plural = _(terms)
                },
                type_facet = {
                        type = fieldList,
                        field = type_id,
```

(continues on next page)

(continued from previous page)

```
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(type),
                        label_plural = _(types)
                },
                status_facet = {
                        type = fieldList,
                        field = status,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(status),
                        label_plural = _(statuses)
                },
                access_facet = {
                        type = fieldList,
                        field = access,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(access status),
                        label_plural = _(access statuses)
                }
        }
}

# ---------------------------------------------------------------------
# Configuration for place browse
ca_places = {
        facets = {
                entity_facet = {
                        # 'type' can equal authority, attribute, fieldList,␣
↪normalizedDates
                        type = authority,
                        table = ca_entities,
                        relationship_table = ca_entities_x_places,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [surname, forname],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(entity),
                        label_plural = _(entities)
                },
                object_facet = {
                        type = authority,
                        table = ca_objects,
                        relationship_table = ca_objects_x_places,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,
```

(continues on next page)

(continued from previous page)

```
                        indefinite_article = a,
                        label_singular = _(object),
                        label_plural = _(objects)
                },
                occurrence_facet = {
                        type = authority,
                        table = ca_occurrences,
                        relationship_table = ca_places_x_occurrences,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(occurrence),
                        label_plural = _(occurrences)
                },
                term_facet = {
                        type = authority,
                        table = ca_list_items,
                        relationship_table = ca_places_x_vocabulary_terms,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(term),
                        label_plural = _(terms)
                },
                type_facet = {
                        type = fieldList,
                        field = type_id,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(type),
                        label_plural = _(types)
                },
                status_facet = {
                        type = fieldList,
                        field = status,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(status),
                        label_plural = _(statuses)
                },
                access_facet = {
                        type = fieldList,
                        field = access,
                        sort_by = [name],
                        group_mode = alphabetical,
```

(continues on next page)

(continued from previous page)

```
                        indefinite_article = an,
                        label_singular = _(access status),
                        label_plural = _(access statuses)
                }
        }
}
# --------------------------------------------------------------------
# Configuration for occurrence browse
ca_occurrences = {
        facets = {
                entity_facet = {
                        # 'type' can equal authority, attribute, fieldList,␣
↪normalizedDates
                        type = authority,
                        table = ca_entities,
                        type_restrictions = [exhibitions],   # if browse for␣
↪occurrences is type-restricted then only display this facet when browsing for␣
↪exhibitions

                        relationship_table = ca_entities_x_occurrences,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [surname, forname],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(entity),
                        label_plural = _(entities)
                },
                object_facet = {
                        type = authority,
                        table = ca_objects,
                        relationship_table = ca_objects_x_occurrences,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(object),
                        label_plural = _(objects)
                },
                term_facet = {
                        type = authority,
                        table = ca_list_items,
                        relationship_table = ca_occurrences_x_vocabulary_terms,
                        restrict_to_types = [],
                        restrict_to_relationship_types = [],
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(term),
                        label_plural = _(terms)
                },
                type_facet = {
                        type = fieldList,
```

(continues on next page)

(continued from previous page)

```
                        field = type_id,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(type),
                        label_plural = _(types)
                },
                status_facet = {
                        type = fieldList,
                        field = status,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(status),
                        label_plural = _(statuses)
                },
                access_facet = {
                        type = fieldList,
                        field = access,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = an,
                        label_singular = _(access status),
                        label_plural = _(access statuses)
                }
        }
}

# ---------------------------------------------------------------------
# Configuration for storage location browse
ca_storage_locations = {
        facets = {
                type_facet = {
                        type = fieldList,
                        field = type_id,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(type),
                        label_plural = _(types)
                },
                status_facet = {
                        type = fieldList,
                        field = status,
                        sort_by = [name],
                        group_mode = alphabetical,

                        indefinite_article = a,
                        label_singular = _(status),
                        label_plural = _(statuses)
                }
        }
}
```

### 1.18.3 Global.conf

The global.conf file is a special configuration file that can be used to define values for substitution into other configuration files. It provides a central location to define sets of values shared across multiple configuration files. All configuration files in the same directory share a common global.conf file.

When a configuration file is loaded, CA looks first for a file named global.conf in the same directory as the requested file. If it finds one, it loads global.conf before any other file. This means that any values defined in global.conf are available for substitution in all configuration files located in the same directory. It also means that any value, scalar or not, in global.conf is available to CA in all configuration files.

### 1.18.4 Media_processing.conf

The file defines the media processing rules to transform media representations to different media transformations.

It is a standard CollectiveAccess configuration file using the *common configuration syntax*.

CollectiveAccess supports media processing configuration for representations of the following items:

- representations (*ca_object_representations, ca_object_representation_multifiles*)
- attribute values (*ca_attribute_value_multifiles*)
- icons (*ca_icons*)
- user comments media (*ca_item_comments_media*)
- representation annotation previews (*ca_representation_annotation_previews*)

You may specify accepted media types, and different versions for the transformations, using rules.

### Common configuration

| Top level key | Description | Default |
|---|---|---|
| use_external_url_when_available | If you want original media fetched from URLs to *NOT* be stored in CA, but rather for CA to directly reference the media via the URL used to fetch it set use_external_url_when_available to 1. If you have no idea what this means then leave this set to zero. | 0 |
| queue_threshold_in_bytes | Filesize (in bytes) above which media should be queued for background processing Files smaller than the threshold will be processed at the time of upload, so you should set this to a small enough value that your server has a shot at processing the media in near-realtime. A safe bet is 500,000 bytes (eg. 0.5 megabytes), but you may need to go lower (or higher).<br>Note that you can override this setting for specific media types and versions below if you wish. Also keep in mind a few other fun facts:<br>1. If the queue_enabled setting in global.conf is set to zero then no background processing will take place, no matter what you set here.<br>2. The default setting for queue_enabled is zero, so make sure you change it if you want background processing to happen.<br>3. Versions that have no QUEUE_WHEN_FILE_LARGER_THAN are never queued for background processing; versions with a QUEUE_WHEN_FILE_LARGER_THAN settings of zero are similarly never queued (absence and zero are one and the same, config-wise).<br>4. Some types of media are setup by default to never queue no matter the “queue_threshold_in_bytes” and “queue_enabled” settings. This includes media types for much little or no processing is done, including SWF, XML and MSWord. | 1000 |

### Organization

At the top level, media_processing.conf is structured as a series of blocks, one for each type of item to be processed:

- representations (*ca_object_representations, ca_object_representation_multifiles*)
- attribute values (*ca_attribute_value_multifiles*)
- icons (*ca_icons*)
- user comments media (*ca_item_comments_media*)
- representation annotation previews (*ca_representation_annotation_previews*)

For each table, there is an associative array, with the following keys:

- *MEDIA_ACCEPT*: Relates mimetypes and media types. Each type of media (Ex. `image`) may have multiple mimetypes associated with it.
- *MEDIA_TYPES*: Describes queueing and available representation versions in different sizes and flavours.
- *MEDIA_TRANSFORMATION_RULES*: Describes the rules to apply to transform a representation file.

This is an example of a media processing file:

```
ca_object_representations = {
  MEDIA_METADATA = "media_metadata",
  MEDIA_CONTENT = "media_content",

  MEDIA_ACCEPT = {
      image/jpeg      = image,
      image/gif       = image,
      image/png       = image,
      image/tiff      = image,
      image/x-bmp     = image,
      image/x-dcraw   = image,
      image/x-psd     = image,
      image/x-exr     = image,
      image/jp2       = image,

      application/octet-stream = binaryfile
  },
  # ---------------------------------------------------------
  MEDIA_TYPES = {
      image = {
          QUEUE = mediaproc,
          QUEUED_MESSAGE =  _("Image is being processed"),
          QUEUE_USING_VERSION = original,
          VERSIONS = {
              thumbnail = {
                  RULE = rule_thumbnail_image, VOLUME = images,
                  QUEUE_WHEN_FILE_LARGER_THAN = <queue_threshold_in_bytes>
              },
              preview = {
                  RULE = rule_preview_image, VOLUME = images,
                  QUEUE_WHEN_FILE_LARGER_THAN = <queue_threshold_in_bytes>
              },
              original = {
                  RULE = rule_original_image, VOLUME = images,
                  USE_EXTERNAL_URL_WHEN_AVAILABLE = <use_external_url_when_available>
              },
```

(continues on next page)

(continued from previous page)

```
              tilepic = {
                  RULE = rule_tilepic_image, VOLUME = tilepics,
                  QUEUE_WHEN_FILE_LARGER_THAN = <queue_threshold_in_bytes>
              }
          },
          MEDIA_VIEW_DEFAULT_VERSION = tilepic,
          MEDIA_PREVIEW_DEFAULT_VERSION = small
      },
      binaryfile = {
          QUEUE = mediaproc,
          QUEUED_MESSAGE =  _("Image is being processed"),
          QUEUE_USING_VERSION = original,
          VERSIONS = {
              thumbnail = {
                  RULE = rule_thumbnail_image, VOLUME = images, BASIS = large,
                  QUEUE_WHEN_FILE_LARGER_THAN = <queue_threshold_in_bytes>
              },
              preview = {
                  RULE = rule_preview_image, VOLUME = images, BASIS = large,
                  QUEUE_WHEN_FILE_LARGER_THAN = <queue_threshold_in_bytes>
              },
              original        = {
                  RULE = rule_original_image, VOLUME = images,
                  USE_EXTERNAL_URL_WHEN_AVAILABLE = <use_external_url_when_available>
              }
          },
          MEDIA_VIEW_DEFAULT_VERSION = large,
          MEDIA_PREVIEW_DEFAULT_VERSION = small
      }
  },
  MEDIA_TRANSFORMATION_RULES = {
      # ----------------------------------------------------------
      # Image rules
      # ----------------------------------------------------------
      rule_thumbnail_image = {
          SCALE = {
              width = 120, height = 120, mode = bounding_box, antialiasing = 0
          },
          SET = {quality = 75, format = image/jpeg}
      },
      rule_preview_image = {
          SCALE = {
              width = 180, height = 180, mode = bounding_box, antialiasing = 0
          },
          SET = {quality = 75, format = image/jpeg}
      },
      rule_original_image = {},
      rule_tilepic_image = {
          SET = {quality = 40, tile_mimetype = image/jpeg, format = image/tilepic}
      }
      # ----------------------------------------------------------
  }
}
```

## MEDIA_ACCEPT

One entry per mimetype. Each type of media (Ex. `image`) may have multiple mimetypes associated with it.

```
mimetype = media_type
```

## MEDIA_TYPES

Each key is a media type descriptor, containing an associative array with queueing and representation version descriptions.

| Key | Description | Example |
|---|---|---|
| QUEUE | Queue to deliver the media type to. Only `mediaproc` is supported currently. | `QUEUE = mediaproc` |
| QUEUED_MESSAGE | Message to show on queue listing | `QUEUED_MESSAGE = _('Image is being processed')` |
| QUEUE_USING_VERSION | Version to use when queueing. Note here that you have to specify a version here that is not set to QUEUE, and that you'd almost always want to be using `original` (a.k.a. the uploaded file). | `QUEUE_USING_VERSION = original` |
| MEDIA_VIEW_DEFAULT_VERSION | Name of the media version that should be used as the default for display for the specified mimetype. • This is only a suggestion - it's the version to display in the absence of any overriding value provided by the user. | `MEDIA_VIEW_DEFAULT_VERSION = tilepic` |
| MEDIA_PREVIEW_DEFAULT_VERSION | Default version to display as a preview for the given field based upon the currently loaded row | `MEDIA_PREVIEW_DEFAULT_VERSION = small` |
| VERSIONS | Versions describe different representation versions. See *VERSIONS* for further details | `VERSIONS = { thumbnail = { RULE = rule_thumbnail_image, VOLUME = images, QUEUE_WHEN_FILE_LARGER_THAN = 1000000 }, preview = { RULE = rule_preview_image, VOLUME = images, QUEUE_WHEN_FILE_LARGER_THAN = 1000000 }, original = { RULE = rule_original_image, VOLUME = images, QUEUE_WHEN_FILE_LARGER_THAN = 1000000 }, tilepic = { RULE = rule_tilepic_image, VOLUME = tilepics, QUEUE_WHEN_FILE_LARGER_THAN = 1000000 } }` |

### VERSIONS

Each key is a version descriptor, containing an associative array, with a pointer to media transformation *rules* that help building a new derivative version of a media file.

| Key | Description | Example |
|---|---|---|
| RULE | Rule name | `RULE = rule_thumbnail_image` |
| VOLUME | A volume label from *Media_volumes.conf* file. Files will be stored in/retrieved from this volume. | `VOLUME = images` |
| QUEUE_WHEN_FILE_LARGER_THAN | File size (in bytes) above which media should be queued for background processing. Files smaller than the threshold will be processed at the time of upload, so you should set this to a small enough value that your server has a shot at processing the media in near-realtime. A safe bet is 500,000 bytes (eg. 0.5 megabytes), but you may need to go lower (or higher).<br>Note that you can override this setting for specific media types and versions below if you wish. Also keep in mind a few other fun facts:<br>1. If the queue_enabled setting in global.conf is set to zero then no background processing will take place, no matter what you set here.<br>2. The default setting for queue_enabled is zero, so make sure you change it if you want background processing to happen.<br>3. Versions that have no QUEUE_WHEN_FILE_LARGER_THAN are never queued for background processing; versions with a QUEUE_WHEN_FILE_LARGER_THAN settings of zero are similarly never queued (absence and zero are one and the same, config-wise).<br>4. Some types of media are setup by default to never queue no matter the “queue_threshold_in_bytes” and “queue_enabled” settings. This includes media types for much little or no processing is done, including SWF, XML and MSWord. | `QUEUE_WHEN_FILE_LARGER_THAN = 1000` |

### MEDIA_TRANSFORMATION_RULES

Rules that describe how to build a derivative version of a media file. There are *operations* on the image and also *filters*.

It is an associative array of operation keys.

Here it is a listing of available **operations**:

| Operation | Description | Example |
|---|---|---|
| ANNOTATE | Add annotations to a media file. Note that annotation requires an alpha channel. If none is available, an all opaque alpha channel is implicitedly created. Not available for GD image plugin.<br>Parameters are:<br>• position: a list of values from `north`, `north_west`, `north_east`, `south`, `south_east`, `south_west`, `center`.<br>• text: the annotation text. You should escape single quote chars in the text.<br>• inset: position of text inside the frame.<br>• font: set the font of the text. Available fonts vary from system to system.<br>• size: Point size for the font.<br>• color: color for the background. accepts a color name, a hex color, or a numerical RGB, RGBA, HSL, HSLA, CMYK, or CMYKA specification, for example, `blue`, `#ddddff`, `rgb(255,255, 255)`. | `ANNOTATE {`<br>`  position = south_east,`<br>`  text = "Annotation",`<br>`  inset = 10,`<br>`  font = Arial,`<br>`  size = 16,`<br>`  color = blue`<br>`}` |
| CROP | Crop the file. Params:<br>• `width`: target width of the file.<br>• `height`: target height of the file.<br>• `x`: horizontal offset.<br>• `y`: vertical offset. | `CROP {`<br>`  width = 100,`<br>`  height = 100,`<br>`  x = 10,`<br>`  y = 10`<br>`}` |
| FLIP | Reflect the scanlines in the vertical or horizontal direction. The image will be mirrored upside-down or left-right. Set direction to `vertical` or `horizontal` | `FLIP { direction =␣`<br>`↪vertical }` |
| ROTATE | Rotate the file. Parameters:<br>• `angle`: angle in degrees. | `ROTATE { angle = 30 }` |
| SCALE | Scale the file. Configure the following params:<br>• `antialiasing`: boolean to activate anti-aliasing.<br>• `width`: new width of the file. It is optional, but one of height or width must be provided.<br>• `height`: new height of the file. It is optional, but one of height or width must be provided. | `# Limit higher dimension␣`<br>`↪to 240px`<br>`SCALE = {`<br>`  width = 240,`<br>`  height = 240,`<br>`  mode = bounding_box,`<br>`  antialiasing = 0`<br>`}`<br><br>`# 200px Square thumbnails`<br>`SCALE = {`<br>`  width = 200,` |

Here it is a listing of available **filters**.

| Filter | Description | Example |
|---|---|---|
| DESPECKLE | Reduce the speckles within an image. No parameters. | `DESPECKLE { }` |
| MEDIAN | Apply a median filter to the image, of the given radius. | `MEDIAN { radius = 2 }` |
| SHARPEN | Use a Gaussian operator of the given radius and standard deviation (sigma). For reasonable results, radius should be larger than sigma. Use a radius of 0 to have the method select a suitable radius.<br>The parameters are:<br>• **radius**: The radius of the Gaussian, in pixels, not counting the center pixel (default 0).<br>• **sigma**: The standard deviation of the Gaussian, in pixels (default 1.0). It can be any floating point value from 0.1, for practically no sharpening, to<br>3 or more for sever sharpening. 0.5 to 1.0 is rather good.<br>The larger the sigma the more it sharpens.<br>• `sharpen 0x.4`: very small<br>• `sharpen 0x1.0`: about one pixel size sharpen<br>• `sharpen 0x3.0`: probably getting too large | `SHARPEN {`<br>`  radius = 0,`<br>`  sigma = 0.63`<br>`}` |
| UNSHARPEN_MASK | This filter sharpens an image. The image is convolved with a Gaussian operator of the given radius and standard deviation (sigma). For reasonable results, radius should be larger than sigma. Use a radius of 0 to have the method select a suitable radius.<br>The parameters are:<br>• **radius**: The radius of the Gaussian, in pixels, not counting the center pixel (default 0).<br>• **sigma**: The standard deviation of the Gaussian, in pixels (default 1.0).<br>• **amount**: The fraction of the difference between the original and the blur image that is added back into the original (default 1.0).<br>• **threshold**: The threshold, as a fraction of QuantumRange, needed to apply the difference amount (default 0.05). | `UNSHARPEN_MASK {`<br>`  radius = 0,`<br>`  sigma = 0.45,`<br>`  amount = 1.5,`<br>`  threshold = 0`<br>`}` |

### 1.18.5 Multipart numbering

The MultiPartIDNumber plug-in provides a flexible means to generate structured numbering systems such as accession numbers within CollectiveAccess. For most numbering schemes employed by museums and archives it should be possible to configure a convenient user interface and adequate validation rules using only the plug-in and without any custom programming.

The MultiPartIDNumber plug-in requires an ID number *format* for each item in CollectiveAccess that supports ID numbers. A format is composed of *elements* joined together by a *separator*. Each element in a format has settings specified that determine what input is valid for it and how it will behave in the user interface. An ID number is constructed by stringing together the individual elements using separators. By combining various types of elements you can create arbitrarily complex numbering systems.

#### The multipart_id_numbering.conf configuration file

The file defines the formats used by the MultiPartIDNumber plug-in. It is a standard CollectiveAccess configuration file using the *common configuration syntax*.

CollectiveAccess supports ID numbers for the following items:

- objects (*ca_objects*)
- lots (*ca_object_lots*)
- entities (*ca_entities*)
- places (*ca_places*)
- collections (*ca_collections*)
- occurrences (*ca_occurrences*)
- loans (*ca_loans*)
- movements (*ca_movements*)
- storage locations (*ca_storage_locations*)
- representations (*ca_object_representations*)
- list items (*ca_list_items*)
- content managed site pages (*ca_site_pages*)
- tours (*ca_tours*)
- tour stops (*ca_tour_stops*)

You may specify numbering formats for any type of item listed above in multipart_id_numbering.conf. The format name for each must be identical to the item code (italicized in the list above). CollectiveAccess will use the format to generate an ID number entry interface and validate input for the ID number field of the respective data item. For lots this is the ‘idno_stub’ field; for objects and other items it is the ‘idno’ field.

The following sample multipart_id_numbering.conf configuration is for an organization employing a lot numbering scheme based upon the year of acquisition and an automatically assigned incrementing lot number. Object numbers are based upon the lot number format but with an additional automatically assigned incrementing item number. In both number formats, elements are separated with periods (“.”). The file specifies a two part number for entities: the first element is a code taken from a drop-down list of three allowable values and the second element is an automatically assigned incrementing number. For both place names and vocabulary terms an automatically assigned incrementing number is specified.

```
formats = {
    ca_objects = {
    # This is a default numbering format for object type for which a format
    # has not been explicitly specified
        __default__ = {
            separator = .,
            # sorting of id numbers will be in reverse of display order
            # (eg. if display is 2011.52.1, sort will be on 1.52.2001);
            # remove sort_order altogether if you want sort to consider
            # elements in display order

            sort_order = [item_num, lot_num, acc_year],

            elements = {
                acc_year = {
                    type = YEAR,
                    width = 6,
                    description = Year,

                    editable = 1
                },
                lot_num = {
                    type = NUMERIC,
                    width = 6,
                    description = Lot number,

                    editable = 1
                },
                item_num = {
                    type = SERIAL,
                    width = 6,
                    description = Item number,

                    editable = 1
                }
            }
        },
        # Here's a specialized number format for objects of type "video"
        # (where "video" is the idno of the object_type)
        video = {
            separator = .,

            elements = {
                acc_year = {
                    type = YEAR,
                    width = 6,
                    description = Year,

                    editable = 1
                },
                typecode = {
                    type = LIST,
                    values = [8MM, DV, BETASP],
                    default = ORG,
                    width = 6,
                    description = Type code,
                    editable = 1
```

(continues on next page)

(continued from previous page)

```
            },
            item_num = {
                type = SERIAL,
                width = 6,
                description = Item number,

                editable = 1
            }
        }
    }
},

ca_object_lots = {
    __default__ = {
        separator = .,

        elements = {
            acc_year = {
                type = YEAR,
                width = 6,
                description = Year,

                editable = 1
            },
            lot_num = {
                type = SERIAL,
                width = 6,
                description = Lot number,

                editable = 1
            }
        }
    }
},

ca_entities = {
    __default__ = {
        separator = .,

        elements = {
            code = {
                type = LIST,
                values = [PER, ORG, GRP],
                default = ORG,
                width = 6,
                description = Entity code,
                editable = 1
            },
            num = {
                type = SERIAL,
                width = 8,
                description = Entity number,
                editable = 1
            }
        }
    }
},
```

(continues on next page)

(continued from previous page)

```
    ca_places = {
        __default__ = {
            # Note the blank separator -- the comma is part of the config
            # file, not the separator value
            separator = ,

            elements = {
                num = {
                    type = SERIAL,
                    width = 8,
                    description = Place number,
                    editable = 0
                }
            }
        }
    },

    ca_collections = {
        __default__ = {
            # Note the blank separator -- the comma is part of the config
            # file, not the separator value
            separator = ,

            elements = {
                num = {
                    type = SERIAL,
                    width = 8,
                    description = Collection number,
                    editable = 0
                }
            }
        }
    },

    ca_occurrences = {
        __default__ = {
            # Note the blank separator -- the comma is part of the config
            # file, not the separator value
            separator = ,

            elements = {
                num = {
                    type = SERIAL,
                    width = 8,
                    description = ID number,
                    editable = 0
                }
            }
        }
    }
}
```

All formats in the configuration file are located in an associative list named *formats* The keys of this list are table names for which format are specified. Each table name key has as its value an associative list keyed on type. Use the special ***__default__*** type to specify a format for use with any type not declared with a specific format.

Each type key has as its value an associative list specifying the format. The following keys may be placed in the list:

| Key | Description | Example value | Mandatory? |
|---|---|---|---|
| separator | The value to be displayed between elements in the user interface for the format. If you want your elements to be merged end to end with no space or separator character(s) then leave this blank. | `separator = ,` | Yes |
| dont_inherit_from_parent_collection | In some archival configurations of CollectiveAccess a cross-table hierarchy is used to link object-level records to the collections they are a part of. By default these child records inherit their parent id number. This is often desired behavior. Other times, for example when a SERIAL configuration is set for object idnos, it's not and has potential to create duplicate id numbers. In these cases dont_inherit_from_parent_collection can be used to prevent object children from inheriting collection idnos that are duplicative. Note this only impacts cross-table hierarchies (doesn't impact other relationships or hierarchies within a single table). | `dont_inherit_from_parent_collection = 1` | No |
| elements | An associative list of elements and the parameters for each. Elements will be output when constructing an ID number or user interface in the same order they appear in the list. At least one element must be declared for the format to be valid. | `elements = {`<br>`    code = {`<br>`        type = LIST,`<br>`        values = [PER, ORG, GRP],`<br>`        default = ORG,`<br>`        width = 6,`<br>`        description = Entity code,`<br>`        editable = 1`<br>`    },`<br>`    num = {`<br>`        type = SERIAL,`<br>`        width = 8,`<br>`        description = Entity number,`<br>`        editable = 1`<br>`    }`<br>`}` | Yes |
| sort_order | By default an ID number will sort on its constituent elements in the order they are defined in the elements list. If you need to have the elements of an ID number display in one order but sort in another, you can set the order used for sorting here. The value should be a simple non-associative list with the element keys in the order to use for sorting. If you want | `sort_order = [num, code]` | No |

The keys of the *element* associative list are element names. These names are only used for reference during configuration and to name HTML form elements and are not presented to the user. They should use only alphanumeric characters and underscores. Do not include spaces or punctuation in the names.

The value for each element name in the elements list is another associative list, this one containing a list of settings declaring the characteristics of the element. The most important setting to set for an element is its type which defines the general range of allowable values and user interface behaviors. The plug-in supports the following types:

| Type | Description |
|---|---|
| LIST | Element value must be taken from a predefined list. User interface is drop-down list containing allowable values. |
| SERIAL | If element value is not set, it will be set to a value one greater than the greatest existing value of the ID number as formed from the element and all preceeding elements in the format. For example, if for the three element format with last last element set to SERIAL 2006 001 001 exists and you enter 2006 4, the last element will be set to 5. It is also possible to set the element value manually, but only letters and numbers are allowed. |
| CONSTANT | Element is always set to a constant alphanumeric value and cannot be changed. |
| FREE | Any input is allowed up to a specified length. |
| NUMERIC | Only numbers are allowed. |
| ALPHANUMERIC | Only numbers and letters are allowed. |
| YEAR | Only valid four digit years are allowed.If empty the element will default to the current year. This is useful when your numbering system includes the current year. |
| MONTH | Only valid month numbers (between 1 and 12) are allowed. If empty the element will default to the current month. |
| DAY | Only valid day numbers (between 1 and 31) are allowed. If empty the element will default to the current day. |
| PARENT | Automatically set to the identifier value for the parent when a new record is created. This type of element is useful when implementing “agglutinative” numbering systems which each level in a hierarchy incorporates the number of its parent. When used this must be the first element in the element list for a format type. Unexpected behaviors may occur if placed in other locations in the element list. |

Beyond type, there are a number of other settings that can be set for an element. Some are common to all element types and others are specific to certain types.

Settings applicable to all types of elements are:

| Parameter | Description |
|---|---|
| description | A short description of the element, suitable for display in error messages. You must set this for each element. |
| editable | If set to a non-zero value element is presented in the user interface as editable (except for elements of type CONSTANT which are never editable). If omitted or set to zero elements are only editable when empty in the case of user-entered element types (LIST, FREE, NUMERIC, ALPHANUMERIC) and never editable in auto-filled element types (SERIAL, CONSTANT, YEAR, MONTH and DAY) |
| width | Display width, in characters, of entry field in user interface. |

Type-specific settings are:

| Type | Parameter | Description |
|---|---|---|
| LIST | values | List of allowable values for the element. Used to validate input and generate a drop-down list in the user interface. Note that the value for this parameter is a simple list, not an associative list. Ex. values = [eins, zwei, drei, vier] |
| | default | Default value for element |
| SERIAL | child_only | Set to restrict element for use only on records with a parent. |
| | zeropad_to_length | If set to a non-zero value, the sequence number is left-padded with zeros until its length is equal to the value. For example, if set to 4, the sequence number 17 would be output as 0017. The zero padding length does not affect sequence numbers longer in length than the specified value. Thus, for example, if zeropad_to_length is set to 3, the sequence value 7114 would be output as-is. |
| | sequence_by_type | If set to a non-zero value sequences are calculated by type. That is, each individual record type within the table gets its own number sequence. By default the numbering sequence is shared amongst all types. For most primary types (objects for example) you'll generally want a single set of numbers for all types. Sometimes, most commonly for occurrences, you may prefer that each type have its own sequence. (available from v1.5) |
| | minimum_value | A non-zero number to use as the initial value for the sequence. Ex. if set to 20000 then the first record created with be numbered 20000, the second 20001 and so on. (available from v1.7) |
| CONSTANT | value | Value of constant. This must be specified. |
| FREE | minimum_length | Minimum allowable length, in characters, of element value. |
| | maximum_length | Maximum allowable length, in characters, of element value. |
| | zeropad_to_length | If set to a non-zero value, the sequence number is left-padded with zeros until its length is equal to the value. For example, if set to 4, the sequence number 17 would be output as 0017. The zero padding length does not affect sequence numbers longer in length than the specified value. Thus, for example, if zeropad_to_length is set to 3, the sequence value 7114 would be output as-is. |
| NUMERIC | minimum_length | Minimum allowable length, in characters, of element value. |
| | maximum_length | Maximum allowable length, in characters, of element value. |
| | minimum_value | Minimum allowable numeric value of element value. |
| | maximum_value | Maximum allowable numeric value of element value. |
| ALPHANUMERIC | minimum_length | Minimum allowable length, in characters, of element value. |
| | maximum_length | Maximum allowable length, in characters, of element value. |
| YEAR | force_derived_values_to_current_year | When deriving an identifier from an existing value (typically when duplicating a record, or creating a child record), force the new value to the current year no matter the existing value. |
| MONTH | force_derived_values_to_current_month | When deriving an identifier from an existing value (typically when duplicating a record, or creating a child record), force the new value to the current month no matter the existing value. |
| DAY | force_derived_values_to_current_day | When deriving an identifier from an existing value (typically when duplicating a record, or creating a child record), force the new value to the current day no matter the existing value. |

### Problems with SERIAL elements

To generate unique values for SERIAL elements the plug-in must query your CollectiveAccess database. If the database operation fails you may see the word 'ERR' instead of the expected numeric value. In versions prior to 1.7.9 the underlying database table and fields used to derive the next number in sequence had to be manually configured for each SERIAL element using the *table*, *field* and *sort_field* settings. If you are running an older version and receive an ERR value verify that the table, field and sort_field element settings are set correctly.

The automatically issued SERIAL values should always be one more than the largest extant value in your database. If you are getting values that are less than the maximum try *reloading sort values*, using the option under the administrative *Maintenance* menu or the command line *caUtils* command using the *rebuild-sort-values* option.

## 1.18.6 Search_indexing.conf

The search_indexing.conf file controls which data in your CollectiveAccess database is searchable, and how. Only data elements configured in search_indexing.conf are searchable. Note that configuration of CollectiveAccess' browse system is completely independent from search. It is possible to search on data that are not browse-able, and browse on elements that are not indexed for search.

### Organization

At the top level, search_indexing.conf is structured as a series of blocks, one for each type of item to be indexed:

```
ca_objects = {
   ... indexing configuration for ca_objects records ...
},
ca_entities = {
   ... indexing configuration for ca_entities records ...
},
ca_places = {
   ... indexing configuration for ca_places records ...
},
ca_occurrences = {
   ... indexing configuration for ca_occurrences records ...
},
...
```

Within each block is a sub-block for item fields as well as sub-blocks for related items and access points (aliases and short cuts for selected data elements or groups of elements). Content in related items may be indexed against the item. For example, you may have an object record indexed by its various fields (accession number, condition, appraised value) as well as by content in related entities (name of artist, nationality of artist), places (place of manufacture), storage location, and more. The object will be searchable by any of the fields for which it has been indexed. Indexing for each type of item is configured independently. You may have objects indexed with content taken from related entities, while omitting related object data from entity indexing, for instance.

A typical ca_objects block might look like this:

```
ca_objects = {
        # ------------------------------------
        ca_objects = {
                fields = {
                        _metadata = { },                                    #␣
↪forces indexing of all attributes
                        parent_id = {STORE, DONT_TOKENIZE, DONT_INCLUDE_IN_SEARCH_
↪FORM },
```

(continues on next page)

(continued from previous page)

```
                        source_id = {},
                        lot_id = {},
                        idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 },
                        type_id = { STORE, DONT_TOKENIZE },
                        source_id = { STORE, DONT_TOKENIZE },
                        hier_object_id = { STORE, DONT_TOKENIZE },
                        access = { STORE, DONT_TOKENIZE },
                        status = { STORE, DONT_TOKENIZE },
                        deleted = { STORE, DONT_TOKENIZE },
                        is_deaccessioned = { STORE, DONT_TOKENIZE },
                        deaccession_notes = {},
                        deaccession_date = {},
                        circulation_status_id = { STORE, DONT_TOKENIZE }
                },
                # Index idno's of related objects
                related = {
                        fields = {
                                idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST =␣
↪100 }
                        }
                }
        },
        # ------------------------------------
        ca_object_labels = {
                key = object_id,
                fields = {
                        name = { BOOST = 100, INDEX_ANCESTORS, INDEX_ANCESTORS_START_
↪AT_LEVEL = 0, INDEX_ANCESTORS_MAX_NUMBER_OF_LEVELS = 4, INDEX_ANCESTORS_AS_PATH_
↪WITH_DELIMITER = . },
                        name_sort = { DONT_INCLUDE_IN_SEARCH_FORM },
                        _count = {}
                },
                # Index names of related objects
                related = {
                        fields = {
                                name = { BOOST = 100, INDEX_ANCESTORS, INDEX_
↪ANCESTORS_START_AT_LEVEL = 0, INDEX_ANCESTORS_MAX_NUMBER_OF_LEVELS = 4, INDEX_
↪ANCESTORS_AS_PATH_WITH_DELIMITER = . }
                        }
                }
        },
                # ------------------------------------
        ca_objects_x_entities = {
                key = object_id,
                fields = {
                        _count = { }
                }
        },
        # ------------------------------------
        ca_entities = {
                tables = {
                        entities = [ca_objects_x_entities]
                },
                fields = {
                        idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 },
                        _count = { }
                }
```

(continues on next page)

(continued from previous page)

```
        },
        # -----------------------------------
        ca_entity_labels = {
                tables = {
                        entities = {
                                ca_objects_x_entities = { },
                                ca_entities = {}
                        },
                        annotations = [ca_objects_x_object_representations, ca_object_
↪representations, ca_representation_annotations, ca_representation_annotations_x_
↪entities, ca_entities]
                },
                fields = {
                        entity_id = { DONT_INCLUDE_IN_SEARCH_FORM },
                        displayname = { PRIVATE },
                        forename = {},
                        surname = {},
                        middlename = {}
                }
        },
        # -----------------------------------
        _access_points = {
                label = {
                        fields = [ca_object_labels.name],
                        options = { DONT_INCLUDE_IN_SEARCH_FORM }
                },
                desc = {
                        fields = [ca_objects.description],
                        options = { }
                },
        }
        # -----------------------------------
}
```

This may look a bit intimidating at first, but there are actually only three types of sub-blocks present: indexing configuration for the item itself (the indented ca_objects key immediately following the first ca_objects that defines the block), indexing from related items (the ca_object_labels keys and those referencing other tables that follow) and access point definitions (the _access_points key at the end of the sub-block). These sub-blocks form the core of the configuration, and are discussed in detail below.

### Sub-blocks

To index data elements that are part of the item itself create a sub-block whose key is the table name of the item. For example, when indexing ca_objects records, define the data elements (metadata attributes intrinsic fields, special fields) to be indexed in a sub-block with the key ca_objects. In the example configuration, this block is defined as:

```
ca_objects = {
        fields = {
                _metadata = { },                                        # forces␣
↪indexing of all attributes
                parent_id = {STORE, DONT_TOKENIZE, DONT_INCLUDE_IN_SEARCH_FORM },
                source_id = {},
                lot_id = {},
                idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 },
                type_id = { STORE, DONT_TOKENIZE },
```

(continues on next page)

(continued from previous page)

```
                source_id = { STORE, DONT_TOKENIZE },
                hier_object_id = { STORE, DONT_TOKENIZE },
                access = { STORE, DONT_TOKENIZE },
                status = { STORE, DONT_TOKENIZE },
                deleted = { STORE, DONT_TOKENIZE },
                is_deaccessioned = { STORE, DONT_TOKENIZE },
                deaccession_notes = {},
                deaccession_date = {},
                circulation_status_id = { STORE, DONT_TOKENIZE }
        },
        # Index idno's of related objects
        related = {
                fields = {
                        idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 }
                }
        }
},
```

The actual fields to index are included in a list with the field key. An additional related key is included, defining indexing for objects related to objects. This will be discussed in detail later.

Each intrinsic field (non-repeating fields hardcoded in the CollectiveAccess database schema) to be indexed is listed individually, with options enclosed in the curly brackets ("{}"). For convenience all configurable metadata elements specific to your installation are indexed using the special _metadata field. This obviates the need for you to enumerate each metadata element individually. If you need to not index certain elements, you can specify individual elements to index using keys starting with ca_attribute_ followed by element codes (ex. metadata element "description" would be listed as "ca_attribute_description").

Only data elements listed in this block, or inferred by the _metadata special field, will be indexed.

**Special fields** There are two "special fields" that may be used in the field list. Special fields always start with underscore character.

| Option | Description |
|---|---|
| _metadata | Forces indexing of all metadata elements configured for the item. When indexing of all fields is desired (the typical case) use of the _metadata special field obviates the need to explicitly list all available fields, and to update indexing configuration every time a new metadata element is added. |
| _count | Embeds the number of related rows for a given table in the index. You can specify this for both relationship (ex. ca_objects_x_entities) and primary (ex. ca_entities) tables. The field is named <table_name>.count - for example: object_representations.count for table 'object_representations'. This can be used to find rows that have, or don't have, related rows in a given table.When specified on a primary table (eg. ca_entities, ca_occurrences), counts are indexed in aggregate as well as for each type. For relationship tables (eg. ca_objects_x_entities) counts are indexed in aggregate as well as for each relationship type. For example querying on a specific type or types: ca_entities.count/individual:3 (finds records with exactly three related entities of type "individual") ca_objects_x_entities.count/artist:[2 to 4] (finds objects with between two and four entities related as artist) |

**Field-level options**

A variety of options are available to control how data elements are indexed:

| Option | Description | Example syntax |
|---|---|---|
| STORE | Forces the value to be stored in the index, if possible; this can speed display of the content in a search but may slow down indexing and increases index size | not applicable |
| TOKENIZE | Breaks content into separate values on whitespace characters, such as a spaces or line breaks, or by punctuation characters prior to indexing. This is the default and in general need not be specified. It may be combined with the DONT_TOKENIZE option to index values both as tokenized fragments and as a single "as-is" value. This can be useful when indexing accession numbers and other identifiers. | not applicable |
| DONT_TOKENIZE | Indexes the value as-is, rather than breaking into separate values on whitespace characters, such as a spaces or line breaks, or by punctuation characters. This is useful for values that should not be indexed as text, such as numeric values and accession numbers/identifiers. | not applicable |
| DONT_INCLUDE_IN_SEARCH_FORM | Indicates that the data element should not be includable in user-defined search forms. | not applicable |
| BOOST | A numeric "boost" value for the index field. Higher values will cause search hits on the boosted field to count for more when sorting by relevance. | BOOST = 100 |
| INDEX_AS_IDNO | Causes the value to be indexed with various permutations for flexible retrieval as a record identifier. For example, if this option is used then a search for KA1 would return KA.0001. | not applicable |
| INDEX_AS_MIMETYPE | Causes the value to be indexed as a mime type variations to support flexible retrieval. For example, if this option is used then mime type values of "image/wav" would be index under both the literal mime type and "WAVE Audio". (Available from version 1.7.1) | not applicable |
| INDEX_ANCESTORS | Enables hierarchical indexing for field, assuming it is in an hierarchical table, resulting in all values for this field in records above the subject in the hierarchy being indexing against the subject | not applicable |
| INDEX_ANCESTORS_START_AT_LEVEL | Forces hierarchical indexing to start X levels down from the root. This allows you to omit the very highest, and least selective, levels of the hierarchy when indexing. If omitted indexing starts from the hierarchy root | INDEX_ANCESTORS_START_AT_LEV = 2 |
| INDEX_ANCESTORS_MAX_NUMBER_OF_LEVELS | Sets the maximum number of levels above the subject to be indexed. If omitted all levels of the hierarchy above the subject are indexed | INDEX_ANCESTORS_MAX_NUMBER = 3 |
| INDEX_ANCESTORS_AS_PATH_WITH_DELIMITER | Sets a delimiter to place between each level of the hierarchy prior to indexing the entire hierarchy path above the subject. This is useful when you want to treat the hierarchy path as an identifier | INDEX_ANCESTORS_AS_PATH_WIT = . |
| PRIVATE | Flags indexing for the data element as being only for use by authenticated users and not for public use. Typically Pawtucket front-ends will ignore indexing so flagged. | not applicable |
| COUNT | For metadata elements only. Causes the number of values set for the element in a record to be indexed. This enables searching on records by the number of values in a given field. (Available from version 1.7) | not applicable |

You can set multiple options by separating them with commas. Options taking values should be separated from the value by an equals sign. For example:

```
ca_objects = {
                fields = {

                        idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 },
```

### Indexing related items

Indexing can traverse relationships and include data elements in related items. This allows, for example, an object to be found using the names of entities related to it. Most of the time only immediate relationships will be indexed (eg. related entities indexed to objects via the object-entities relationship), but it is possible to specify any path between items. Thus you could, for example, index entities against objects via a third item, such as occurrences.

Sub-blocks for related items have their key set to the related table name. A typical sub-block, for entity preferred labels on objects, might look like this:

```
ca_entity_labels = {
        tables = {
                entities = [ca_objects_x_entities, ca_entities],
                annotations = [ca_objects_x_object_representations, ca_object_
↪representations, ca_representation_annotations, ca_representation_annotations_x_
↪entities, ca_entities]
        },
        fields = {
                entity_id = { DONT_INCLUDE_IN_SEARCH_FORM },
                displayname = { PRIVATE },
                forename = {},
                surname = {},
                middlename = {}
        }
},
```

We index in the ca_entity_labels table, because that is the table storing entity labels (other times have similarly named tables: ca_object_labels, ca_occurrence_labels, etc.). The tables key specifies the table path(s) through the CollectiveAccess database to use to connect entity labels to objects. In this example we specify two paths, one via ca_objects_x_entities (the direct relationship), and one via object representations and representation annotations. The path should be the sequence of tables to traverse, starting with the table being indexed (in this example, ca_objects), which is omitted.

The fields key includes all fields in the related table that should be indexed. In this case we index four name component fields for entities against objects. The same special fields and options available when indexing fields in the item itself are available when indexing related items.

**Indexing preferred and non preferred labels**

Labels are stored in the CollectiveAccess database as related records, and can be indexed similarly to other related items. One difference: most relationships are many-to-many, with a relationship table in between. Labels are related many-to-one without a relationship table, resulting in a simpler configuration. When indexing labels or any other many-to-one relationship (ex. objects - object lots) you need any specify the name of the field in the related table that references the primary item. In the example below this field name is object_id and is configured using a key

```
ca_object_labels = {
        key = object_id,
        fields = {
                name = { BOOST = 100, INDEX_ANCESTORS, INDEX_ANCESTORS_START_AT_LEVEL␣
↪= 0, INDEX_ANCESTORS_MAX_NUMBER_OF_LEVELS = 4, INDEX_ANCESTORS_AS_PATH_WITH_
↪DELIMITER = . },
```

(continues on next page)

(continued from previous page)

```
                name_sort = { DONT_INCLUDE_IN_SEARCH_FORM },
                _count = {}
        },
        # Index names of related objects
        related = {
                fields = {
                        name = { BOOST = 100, INDEX_ANCESTORS, INDEX_ANCESTORS_START_
↪AT_LEVEL = 0, INDEX_ANCESTORS_MAX_NUMBER_OF_LEVELS = 4, INDEX_ANCESTORS_AS_PATH_
↪WITH_DELIMITER = .  }
                }
        }
},
```

**Counting related items**

The number of related items can be indexed using the _count special field. When placed in the sub-block for the related item (ca_entities in our example), counts will be indexed in total and by item type (eg. by entity type). When _count is placed in a sub-block for a relationship table, counts will be indexed in total and by relationship type. In our example this configuration indexes counts for related entities on the object, broken out by relationship type:

```
ca_objects_x_entities = {
        key = object_id,
        fields = {
                _count = { }
        }
}
```

Indexed counts may be searched using the count field on the appropriate table.

**Controlling related indexing**

Indexing of related items can be restricted to specific relationship types using an alternate syntax for the tables list. Rather than using a list of tables:

```

```

you can specify an associative array with additional setting:

```
entities = {
        ca_objects_x_entities = {
                types = [artist, publisher]
        },
        ca_entities = {}
}
```

the types setting is a list of relationship types to restrict indexing to.

You can also flag related indexing in a sub-block as private (not to be used in public interfaces) by specifying the PRIVATE option in relevant table paths.

As of version 1.7.6 it is possible to restrict indexing by related item type using a “types” key in a sub-block with a list of types to restrict to.

**Indexing self-relationships**

“Self-relationships” are connections between two items of the same kind, such as object-to-object and entity-to-entity relationships. Indexing configuration for this sort of relationship is handled differently then that of other related items. To index self-relationships include a “related” key in the sub-block for the item table. In our example the block is:

```
ca_objects = {
                fields = {
                        _metadata = { },                                        #␣
↪forces indexing of all attributes
                        parent_id = {STORE, DONT_TOKENIZE, DONT_INCLUDE_IN_SEARCH_
↪FORM },
                        source_id = {},
                        lot_id = {},
                        idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 },
                        type_id = { STORE, DONT_TOKENIZE },
                        source_id = { STORE, DONT_TOKENIZE },
                        hier_object_id = { STORE, DONT_TOKENIZE },
                        access = { STORE, DONT_TOKENIZE },
                        status = { STORE, DONT_TOKENIZE },
                        deleted = { STORE, DONT_TOKENIZE },
                        is_deaccessioned = { STORE, DONT_TOKENIZE },
                        deaccession_notes = {},
                        deaccession_date = {},
                        circulation_status_id = { STORE, DONT_TOKENIZE }
                },
                # Index idno's of related objects
                related = {
                        fields = {
                                idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST =␣
↪100 }
                        }
                }
        },
```

The configuration for the self-relationship indexing is in bold. The fields are configured similarly to other types of indexing, with the same options and special fields. related indexing for preferred and non-preferred labels may be added in the many-to-one label indexing configuration.

As of version 1.7.4 you can also include a list of types to restrict related indexing to. If you wish, for example, to only index related objects of type "artwork" and "book" against other objects the relevant fragment of configuration might look like so:

```
# Index idno's of related objects
related = {
   types = [artwork, book],
   fields = {
      idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 }
   }
}
```

**Indirect self-relationships**

As of version 1.7.6 it is also possible to index two items of the same kind indirectly, through a series of relationships to items of other types. For example, objects can be indexed against other objects that share the same related entities. In this case you would be indexing objects > entities > objects.

To configure this sort of indexing create a sub-block with a key set to the item name followed by ".related" You can then configure indexing as you would for any other related record. For objects the configuration might look like this fragment:

```
ca_objects.related = {
        tables = {
                entity = [ca_objects_x_entities, ca_entities, ca_objects_x_entities]
```

(continues on next page)

(continued from previous page)

```
        },
        fields = {
                idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 }
        }
},
```

This would index objects with the idno field values of any objects with the same related entities.

**Access Points**

The access points sub-block (use key _access_points) defines aliases for specific indexed elements or groups of elements. It also allows a user to set attributes to be used in search forms as well as search shortcuts.

**Search Shortcuts**

With _access_points you can create shortcuts to be used in any search system-wide, including Basic Search, Quick Search, Find in the Hierarchy bundle, and Advanced Search.

Let's say you want to create a search shortcut for a "Materials" element on your object record. In the Access points sub-section of the objects section of your configuration file:

```
ca_objects = {
       # ------------------------------------
       _access_points = {
```

you would add the "Materials" access_point. Whatever you want the shortcut to be (let's say "mat") should be included on the left side of the equals sign:

```
ca_objects = {
       # ------------------------------------
       _access_points = {
               mat = {
                       fields = [ca_objects.material],
                       options = { DONT_INCLUDE_IN_SEARCH_FORM }
               },
```

Within the square brackets to the right of the fields equals sign, the attribute's elementCode is used (following a period and the CA table name).

Now you can quickly search for materials anywhere in your system using the syntax:

```
mat:stone
```

It is also possible to create shortcuts that bundle several elements together. A search on the access point will search all of the included fields at the same time. Each attribute should be comma separated:

```
style = {
       fields = [ca_objects.material, ca_objects.medium, ca_objects.technique],
```

Remember that if you want to search for multiple words within your single access point, quotation marks should enclose the whole string:

```
style:"stone sculpture"
```

A search for simply:

```
style:stone sculpture
```

would mean search for stone in the Materials, Medium & Technique fields AND sculpture anywhere else. That would mostly likely also return effective (but different) search results. Similarly, there shouldn't be a space between the colon and the search term (i.e. style: stone) because the search will "break" on the space and the search preformed will be a universal query for stone.

If your target element for a search shortcut is a container, make sure to include the full path of ca_table.elementCode.subElementTarget or:

```
fields = [ca_objects.description.description_source],
```

**Search forms**

You may have noticed that in the code examples above an option was used:

```
options = { DONT_INCLUDE_IN_SEARCH_FORM }
```

This is because by default each defined metadata element will be pulled into the available elements for building search forms. Including your shortcut a second time would be redundant. However, if you're adding an access point that isn't already included (say, "filename" which until recently wasn't indexed by default but was stored in the database) you would define it here and remove the DONT_INCLUDE_IN_SEARCH_FORM option.

Note that all fields included in an access point must be included in the search index - they must appear in the fields list in other words. All indexed fields automatically have access points created in the format tablename.fieldname (ex. objects.title); indexed metadata also have access points in the format tablename.md_<element_id> (ex. objects.md_5)

**Rebuilding the search index**

Changes to search_indexing.conf take effect immediately for all subsequent indexing. Any items indexed prior to the change will not reflect the configuration modifications. To update the entire search index to reflect the new configuration, rebuild the index using "Rebuild search indices" web interface under Manage > Administrate > Maintenance; or reindex using the command-line caUtils rebuild-search-indices command

## 1.18.7 Authentication

Providence (and Pawtucket) allow for a range of authentication providers to be configured. By default this is handled internally, but external providers can be used, including federation authentication services.

## 1.18.8 Datetime.conf

''In Progress''

The output, or display, of dates and times inside dateRange metadata elements can be configured in datetime.conf. For valid <em>input</em> formats for dates and times, please visit [[Date_and_Time_Formats|this page.]]

### Date/time output configuration

In Datetime.conf, you may define common text expressions you wish to have the date/time parser convert to dates. The text expression on the left side of the equal sign must be *all lowercase*; the date/time expression on the right side must be valid and parsable:

```
expressions = {
   us civil war = 1861 to 1865,
       world war 2  = 1939 to 1945,
   nickel empire = 1920s,
}
```

### Output options for date/times

| Setting | Description | Options |
|---|---|---|
| dateFormat | Format to use for dates. "Original" is the date as entered by the user; other values will normalize all date/time input to the selected standard format. | Valid values are text, delimited, iso8601, and original. The default is text. |
| timeOmit | You may output or omit the time portion of date time expressions. | 1 (yes) or 0 (no) |
| showCommaAfterDayForTextDates | If set to a non-zero value commas are included after the day in a US-style (month first) text date | Default = 0 |
| timeFormat | Format to use for times. "12" displays as, for example, 3:15 PM, where "24" would display at 15:15PM. | Valid values are 12 and 24. Default = 24. |
| useQuarterCenturySyntaxForDisplay | If true dates ranging over uniform quarter centuries, such as 1900-1925 or 1975-2000 will be output in the format "20 Q1" eg. 1st quarter of 20th century, or 1900-1925. | 1 (yes) or 0 (no) |
| useRomanNumeralsForCenturies | If true century only dates (eg "18th century") will be output in roman numerals like "XVIIIth century". | 1 (yes) or 0 (no) |
| timeDelimiter | Delimiter in time display; must be valid for the current language or default will be used; Default is first delimiter in language config file. | : |
| timeRangeConjunction | Text to put between times in a range; must be valid for the current language or default will be used; default is first in language config. | - |
| rangePreConjunction | Text to place before date/times in a range; must be valid for the current language or default will be used. Default is none. | from |
| rangeConjunction | Text to place between date/times in a range; must be valid for the current language or default will be used. | to |
| dateTimeConjunction | Text to put between times in a range; must be valid for the current language or default will be used; default is first in language config. | to |
| showADEra | If set to a non-zero value the "AD" era will be show for all dates; default is to only show it in ranges that span era | 1 or 0 |
| uncertaintyIndicator | Text to use to indicate date is uncertain; must be valid for the current language or default will be used. | circa |
| dateDelimiter | Text to place before date/times in a range; must be valid for the current language or default will be used. Default is none. | |
| circaIndicator | Text to place before date/times to indicate it is a "circa", or uncertain, date. Must be valid for the current language or default will be used. | circa |
| beforeQualifier | Text to place before a date/time to indicate that it is no later | before/prior to |

### 1.18.9 Dimensions.conf

### 1.18.10 External_applications.conf

Several components of CollectiveAccess employ external applications to perform various tasks. Typically these tasks relate to conversion and reformatting of uploaded media (images, video, audio, etc.) and indexing of text embedded in uploaded files.

The *external_applications.conf* file defines the locations of these applications on your server. If an application location is set incorrectly or the application is not installed then the functionality provided by the application will not be available within CollectiveAccess.

The locations you set should be absolute paths to the directory or executable (as specified below) in the standard format for your OS (Unix paths or Windows paths).

#### Directives

The following entries may be defined in this configuration file. Note that there are no default values for entries in *external_applications.conf*. You must define a value for all applications you wish to use.

| Entry | Description | Typical value (not default: just an example in Unix path format) |
|---|---|---|
| ghostscript_app | Path to Ghostscript binary ("gs" command) used to generate page images from PDF files | /usr/local/bin/gs |
| ffmpeg_app | Path to ffmpeg binary used to convert video and audio media | /usr/local/bin/ffmpeg |
| qt-faststart_app | Path to [ttp://ffmpeg.mplayerhq.hu qt-faststart] binary used to hint h.264/MPEG-4 video for streaming. qt-faststart is part of ffmpeg and located in the tools/ directory in the source tree. | /usr/local/bin/qt-faststart |
| dcraw_app | Path to dcraw binary used to convert various proprietary RAW formats produced by digital cameras | /usr/bin/dcraw |
| imagemagick_path | Path to directory containing ImageMagick binaries used to convert various image formats. Note that unlike the other entries in this file, imagemagick_path refers to a directory rather than a specific executable | /usr/local/bin |
| pdftotext_app | Path to pdftotext binary (part of the xpdf package from) used to extract text embedded in PDF files | /usr/local/bin/pdftotext |
| media_info_app | Path to MediaInfo binary used to extract metadata from media files. MediaInfo is optional and is used if present because it generally does a better job of extracting metadata than the methods built into CA and its media processing plugins. | /usr/local/bin/mediainfo |
| libreoffice_app | Path to LibreOffice 3.6 or better binary used to process uploaded Microsoft Word, Excel and PowerPoint files. If you wish to have Word, Excel and PowerPoint content indexed for search and previews generated, you must have LibreOffice installed. See this cookbook entry for more information. | /usr/bin/libreoffice |
| pdfminer_app | Path to the directory containing the PDFMiner script used to analyze uploaded PDF files. Like PDFToText, PDFMiner can extract text for indexing from PDF files. It can also extract page locations for individual words, enabling search-within-PDF functionality in CollectiveAccess. If you want to search within PDFs with term highlighting in CA PDFMiner must be installed. | /usr/local/bin |
| exiftool_app | Path to the ExifTool application (http://www.sno.phy.queensu.ca/~phil/exiftool/) used to extract metadata from uploaded media files. | /usr/bin/exiftool |
| phantomjs_app | Path to the PhantomJS HTML-to-PDF converter (http://phantomjs.org) used to generate printable PDF output. | /usr/local/bin/phantomjs |
| wkhtmltopdf_app | Path to the wkhtmltopdf HTML-to-PDF converter (http://wkhtmltopdf.org) used to generate printable PDF output. | /usr/local/bin/wkhtmltopdf |
| openctm_app | Path to the OpenCTM (http://openctm.sourceforge.net) ctmconv command line utility used to compress PLY and STL 3d models for high efficient delivery and in-browser display. (From version 1.6) | /usr/local/bin/ctmconv |
| meshlabserver_app | Path to the MeshLab (http://meshlab.sourceforge.net) meshlabserver command-line utility, used to convert PLY files to STL. (From version 1.6) | /usr/local/bin/meshlabserver |

### 1.18.11 Library_services.conf

### 1.18.12 OAI_harvester.conf

### 1.18.13 OAI_provider.conf

### 1.18.14 Prepopulate.conf

The Prepopulate plugin provides a system for automatically setting data in records during editing using *display templates* and *Expressions*. Common use cases include:

- Replicating data between parent and child records in hierarchies or related records.
- Generating formatted text values using several metadata values and, potentially, context-specific logic. (Formatting various field values into a bibliographic citation, for example)
- Forcing one or more fields to default values based upon the value of another field.

Prepopulate applies configured rules to records as they are edited. Rule typically define a `target` to set a value for and a *display template* with which to generate the value. Templates are evaluated relative to the record being edited so all values accessible with respect to that record, including related records and parent and child records in hierarchies are available to the template. Rules may be constrained to apply to specific tables and, optionally, record types. Application of rules can be made contingent upon evaluation of an *Expressions* (which can also reference all values accessible to the edited record) or the current status of the target metadata element.

It is also possible to configure rules that replicate relationships between records. As of version 1.7.9, Prepopulate may be configured to replicate between records container metadata elements in whole or in part.

#### Basic Setup

All configuration is made in the `prepopulate.conf` configuration file. The `enabled` directive governs whether Prepopulate is active or not. It must be set to a non-zero value for any Prepopulate-based actions to occur. Two other directives control which user actions trigger application of configured rules when Prepopulate is enabled:

- `prepopulate_fields_on_edit` will cause rules to be applied whenever a record is opened in the editor.
- `prepopulate_fields_on_save` will trigger application of rules whenever a record is saved.

For rules to be applied `enabled` and at least one of `prepopulate_fields_on_edit` and `prepopulate_fields_on_save` must be set.

The `prepopulate_rules` directive contains a dictionary of rules to apply. Each key is a unique alphanumeric identifier for a rule. The precise value is not critical, but it must be unique and should be meaningful. Corresponding values are dictionaries with keys and values defining rule behavior.

An example `prepopulate_rules` dictionary with a single rule with code `test_rule` is shown below:

```
prepopulate_rules = {
   # --------------------
   test_rule = {
      # what types of records does this rule apply to?
      table = ca_objects,
      restrictToTypes = [artwork],

      # mode determines handling of existing values in target element
      # can be overwrite, or addIfEmpty
      # See the 'target' setting below
      mode = addIfEmpty,
```

(continues on next page)

(continued from previous page)

```
      # What's the prepopulate target?
      # This can be an intrinsic field, labels or an attribute.
      #
      # Note that if you want to target a List attribute, you have to
      # provide a valid list item idno or id for that list as value!
      #
      target = ca_objects.title_notes,

      # skip this rule if expression returns true
      # available variable names are bundle names
      skipIfExpression = ^ca_objects.idno =~ /test/,

      # content to prepopulate
      # (this is a display template evaluated against the current record)
      template = ^ca_objects.preferred_labels (^ca_objects.idno),
   }
       # -------------------
}
```

This rule applies to object records (see the `table` setting) of type "artwork" (see `restrictToTypes`). When evaluated, it will fill the "title_notes" field (see `target`) with the object record's preferred label and identifier, formatted with the identifier in parens (see `template`). The rule will be skipped if the object identifier contains the word "test" (see `skipIfExpression`) or of there is already a value in the "title_notes" field for the object (see `mode`).

### Replicating relationships and containers

Most rules generate a text value using `template` and copy it to the `target`, subject to optional restrictions (`mode`, `skipIfExpression`, `restrictToTypes`, etc). It is also possible to replicate relationships in Prepopulate using the `context` directive. In this case, `target` is the type of relationship to replicate and `context` defines the source of the relationship. Possible contexts are "related", "parent" or "children".

An example configuration for replicating relationships using `context` follows:

```
related_entities = {
   table = ca_objects,

   # add relationships that do not already exist
   mode = merge,

   # copy all entities related to objects related to the target record
   target = ca_entities,
   context = related,

   # copy only those entities related with the relationship type "artist"
   restrictToRelationshipTypes = [artist],

   # don't copy relationships with specified relationship type codes;
   #excludeRelationshipTypes = [],

   # copy only entities that are the type "individual"
   restrictToRelatedTypes = [individual],

   # don't copy relationships pointing to specified types
   #excludeRelatedTypes = [],
```

(continues on next page)

(continued from previous page)

```
    # only consider "current" relationships - Eg. current storage location
    currentOnly = 0,
}
```

The example above copies all entity relationships to entities of type "individual" on objects *related* to the currently edited object. If the context had been set to "parent" entity relationships on the parent object would have been copied to the currently edited object.

Individual values in a container metadata element can be copied using the standard `template`/`target` rules described earlier. To copy an entire container between records without requiring a separate rule for each sub-element use the `source` directive to specify the container you wish to copy to the `target`. Prepopulate will assume the `source` and `target` containers have identical structure. To map values between different structures use the `sourceMap` directive to create a conversion table mapping equivalent sub-elements in each container.

An example configuration for replicating container values in their entirety from a parent record to a child record using `source` and `sourceMap` is below:

```
dimensions_container_rule = {
                table = ca_objects,
                restrictToTypes = [edition_item],

                mode = addIfEmpty,

                target = ca_objects.edition_dimensions,

                # skip this rule if expression returns true
                # available variable names are bundle names
                #skipIfExpression = ^ca_objects.idno =~ /test/,

                # for prepopulation of full containers where the container has the␣
↪same
                # format in both the source and target you can copy it directly by␣
↪specifying
                # a "source" specification. Sub-element codes must match exactly for␣
↪this to work.
                source = ca_objects.parent.edition_dimensions,

                # If sub-element codes don't match exactly you can specify a mapping␣
↪where source
                # keys are on the left and target keys on the right. This also␣
↪enables partial copy
                # of containers, as when sourceMap is specified only those keys␣
↪defined in the map are copied
                sourceMap = {
                        edition_display_dimensions = edition_display_dimensions,
                        edition_dimensions_height = edition_dimensions_height,
                        edition_dimensions_width = edition_dimensions_width,
                        edition_dimension_types = edition_dimension_types,
                        edition_dimensions_notes = edition_dimensions_notes
                }
        }
```

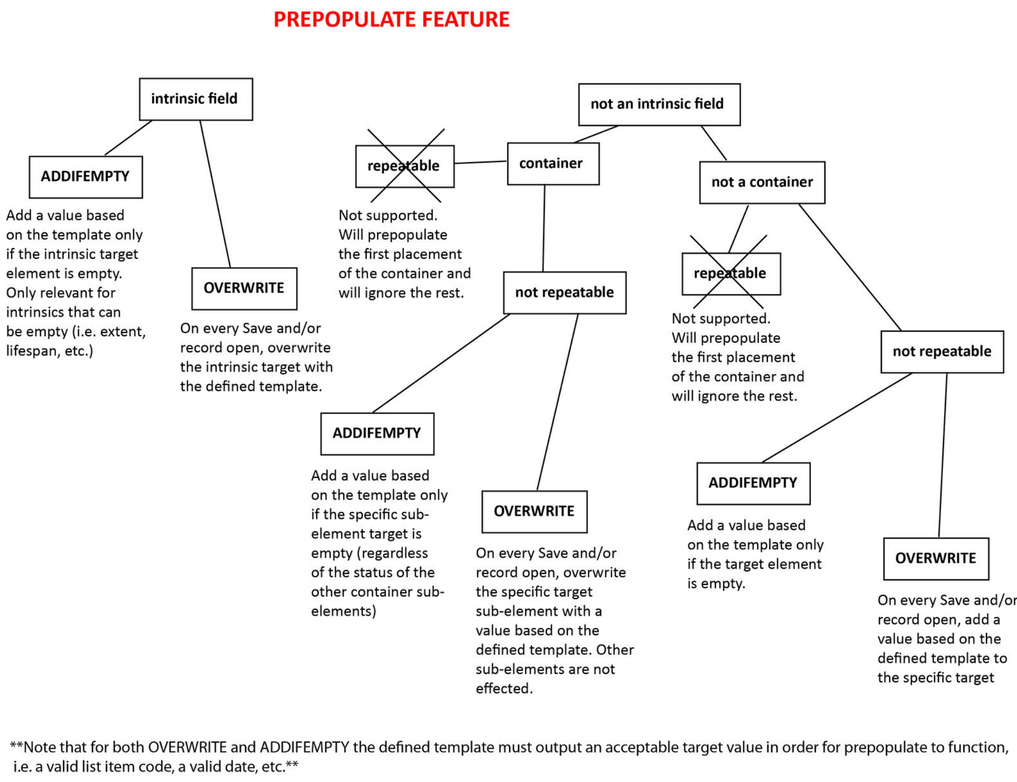
### Settings

The following settings are available when configuring Prepopulate rules:

| Setting name | Description | Valid values | Example |
|---|---|---|---|
| table | Defines what table the rule applies to. Required. | Any *primary table* | ca_objects |
| restrictToTypes | Optional list of types to restrict rule to. | Any valid type for the table. | [artwork, image] |
| mode | Required setting controlling how and if rule is applied when the target contains existing values. See flowchart below. Options are:<br>addIfEmpty: set value only if none already exists<br>overwrite: replace existing values; if the value to be set in the target is empty existing values will be removed but not replaced<br>overwriteifset: replace existing values only if the value to be set to a non-empty value<br>merge: add values that do not already exist | One of addIfEmpty, overwrite, overwriteifset, merge | addIfEmpty |
| target | Specifies the metadata element that the rule will set values for. The target must be part of the record being edited. The value set is defined by the template or source directives.<br>Note that if when targetting a List metadata element, you must provide a valid list item idno or numeric item_id as the value. | A valid bundle specifier for either a intrinsic field, labels or an metadata element | [ca_objects.description] |
| template | A display template defining the value to be set in the target. The display template is evaluated against the current record and can incorporate any value accessible to the currently edited record, including related records and hierarchical values. Required, unless the rule is targeting a container metadata element with a full copy of another container in which case `source` must be set. | A valid *display template*. | ^ca_objects.medium_container.medium ^ca_objects.medium_container.support |
| source | Rules using `template` set only a single text value in the target. When prepopulating container metadata elements with values another instance of the container in a related record (Ex. replicating a container in a parent record to a child record) `source` may be used to specify the instance to copy from. Sub-element codes are assumed to match exactly, so on its own `source` is only useful for copying containers from related or hierarchical parent or child records. It is possible to selectively copy elements from different containers using source in conjunction with the `sourceMap` directive.. Available from version 1.7.9. | A valid bundle specifier for a container metadata element. | ca_objects.dimensions |
| sourceMap | A dictionary mapping sub-element codes in a source container (specified by the `source` directive) to sub-element codes in the target container. Only values defined in the dictionary will be copied to the target. SourceMap can be used to copy values betwen differently structured containers or, by omitting sub-elements, to selectively copy data between two instances of the same container. Available from version 1.7.9. | A dictionary container keys set to source container sub-element codes and values set to target sub- | { edition_display_dimensions = edition_display_dimensions, edition_dimensions_height = edition_dimensions_height, edition_dimensions_width = edition_dimensions_width, edition_dimension_types = edition_dimension_types, |

### Flowchart

Typical Prepopulate processes are diagrammed below. Note that the mode "overwriteifset" (which is not shown in the diagram) is identical to "overwrite" save that no overwrite is performed for empty values.


PREPOPULATE FEATURE
intrinsic field
ADDIFEMPTY
Add a value based on the template only if the intrinsic target element is empty. Only relevant for intrinsics that can be empty (i.e. extent, lifespan, etc.)
OVERWRITE
On every Save and/or record open, overwrite the intrinsic target with the defined template.
repeatable
Not supported. Will prepopulate the first placement of the container and will ignore the rest.
not an intrinsic field
container
not repeatable
ADDIFEMPTY
Add a value based on the template only if the specific sub-element target is empty (regardless of the status of the other container sub-elements)
OVERWRITE
On every Save and/or record open, overwrite the specific target sub-element with a value based on the defined template. Other sub-elements are not effected.
not a container
repeatable
Not supported. Will prepopulate the first placement of the container and will ignore the rest.
not repeatable
ADDIFEMPTY
Add a value based on the template only if the target element is empty.
OVERWRITE
On every Save and/or record open, add a value based on the defined template to the specific target
**Note that for both OVERWRITE and ADDIFEMPTY the defined template must output an acceptable target value in order for prepopulate to function, i.e. a valid list item code, a valid date, etc.**

## 1.18.15 Search.conf

### Indexing Tokenizer Regex

This is the Regex character class used when indexing saved text; values matched will be used as token delimiters (in other words, the search expression will be broken into words wherever the matched characters are). Note that the default class, as displayed in the example below, starts with a caret ("^"), which has the effect of negating the class. In other words, the class defines what characters will not be treated a token delimiters.

```
indexing_tokenizer_regex = ^\pL\pN\pNd/_#\@\&\.
```

### Search Tokenizer Regex

This is the Regex character class used when searching; values matched will be used as token delimiters (this is the same thing as indexing_tokenizer_regex except that it's used when to break user searches into words rather than text to be indexed).

```
search_tokenizer_regex = ^\pL\pN\pNd/_#\@\&\.
```

### "As Is" Regex Matching for Accession Numbers

Here you may enter a list of regular expressions that if matched cause search input to be treated "as-is," or searched without being broken up into tokens. This is useful for preventing tokenization of accession numbers and other values that rely upon punctuation being kept intact when being searched.

```
asis_regexes = [
        "^[\d]+[\.\-][A-Za-z0-9\.\-]+$"
]
```

### Changing the layout of quicksearch results

With the following format:

```
ca_<table>_<type>_quicksearch_result_display_template =
```

or

```
ca_<table>_quicksearch_result_display_template =
```

The format of the quick search results can be altered. The value of the template uses the same syntax as bundle displays. The below is an example for adding "artists" to an "artwork" search result layout:

```
ca_objects_artwork_quicksearch_result_display_template =
<unit relativeTo='ca_entities' restrictToRelationshipTypes='artist'><u>^ca_entities.
↪preferred_labels.surname, ^ca_entities.preferred_labels.forename</u>:</unit>
<em>^ca_objects.preferred_labels.name</em> (<l>^ca_objects.idno</l>) [^ca_objects.
↪type_id]
```

### SqlSearch Plugin Configuration

Set to 0 if you don't want search input stemmed (ie. suffixes removed) prior to search

The plugin uses the English Snoball stemmer (http://snowball.tartarus.org/) and can give poor results with non-English content. If you are cataloguing non-English material you will probably want to turn this off.

search_sql_search_do_stemming = 1

### ElasticSearch Plugin Configuration

enter the elastic search base url here (without any index names) search_elasticsearch_base_url = http://localhost:9200/

This is the name of the ElasticSearch index used by CollectiveAccess. You probably don't need to change this unless you're using a single ElasticSearch setup for multiple CollectiveAccess instances and/or other applications. search_elasticsearch_index_name = collectiveaccess

### 1.18.16 Visualization.conf

### 1.18.17 Attribute_presets.conf

### 1.18.18 Default_media_icons.conf

### 1.18.19 External_exports.conf

### 1.18.20 Linked_data.conf

Add table for Getty Information Services from this page: https://docs.collectiveaccess.org/wiki/Information_Services

### 1.18.21 Media_display.conf

The media_display.conf file controls how media representations are displayed in both the media overlay and media editor. Display settings can be customized for images, video, video H264 original, quicktime Viewer, audio, pdf files, documents, postscript, and text.

#### Media Overlay and Media Editor

The media overlay is accessed by clicking through the representation from the Inspector in Providence. The media editor, on the other hand, is accessed by clicking through the thumbnail representation on the media editor itself. You can set the display options differently here, or configure them to be identical to the media overlay.

| Option | Description | Example syntax |
|---|---|---|
| mimetypes | Mimetypes are are codes that unambiguously identify a media format. By default, nearly all supported mimetypes are included, but the user is free to add more according to the capabilities of the server and which plugins are running. | image/jpeg, image/tiff, image/png |
| display_version | Controls which image or video display version is shown in the overlay or editor. | tilepic, video/mp4 |
| viewer_width | Sets the width of the media in the overlay or editor. | 100% |
| viewer_height | Sets the height of the media in the overlay or editor. | 100% |
| use_book_viewer_when_number_of_representations_exceeds | When the number of media representations exceeds the number set here, the book viewer will be used to display the images. | 2 |
| use_book_viewer | Enables the bookviewer. For documents, enabling this in addition the pdfjs viewer will allow non-pdf documents to be shown in the Bookviewer while PDFs will continue to be shown in the pdfjs viewer. | 1 (yes) or 0 (no) |
| show_hierarchy_in_book_viewer | If the record has sub-records with media, media representations of child records will be shown if the hierarchy is enabled. | 1 (yes) or 0 (no) |
| restrict_book_viewer_to_types | You can restrict the use of the book viewer to particular object types by entering the type code for each between the brackets, separating types by comma | [object_type_code, object_type_code] |
| download_version | This sets which version of the media can be downloaded from the media editor. | original, large |
| poster_frame_version | Specifies which version to use for the video player as a still image prior to starting playback. | mediumlarge |
| alt_display_version | Specifies the version of still image to be used when video cannot be displayed. For example, what would be displayed for a Flash video of the user did not have Flash. | large |
| use_pdfjs_viewer | Enables the recommended viewer for pdf files. | 1 (yes) or 0 (no) |

### 1.18.22 Media_metadata.conf

### 1.18.23 Navigation.conf

### 1.18.24 Replication.conf

### 1.18.25 Services.conf

### 1.18.26 Statistics.conf

### 1.18.27 Translations.conf

### 1.18.28 Annotation_types.conf

### 1.18.29 Assets.conf

### 1.18.30 Attribute_types.conf

### 1.18.31 Datamodel.conf

### 1.18.32 Find_volumes.conf

### 1.18.33 Find_navigation.conf

### 1.18.34 Media_volumes.conf

CollectiveAccess stores uploaded media and derivatives in the media directory.

You can change the location where media is stored by editing the media volumes configuration file in `app/conf/media_volumes.conf`.

#### Organization

The file includes an entry per volume.

| Key | Description | Example |
|---|---|---|
| hostname | Hostname | <site_hostname> |
| protocol | Protocol (ftp, http[s], etc.) | <site_protocol> |
| urlPath | Base path for exposing the media on this volume | <ca_media_url_root>/images |
| absolutePath | Absolute filesystem path for the volume | <ca_media_root_dir>/images |
| writeable | A flag to indicate if the volume is writeable or not | 1 |
| description | Description of the volume | Images |
| accessUsingMirror | Name of mirror to serve media from, when available. If the specified mirror does not yet have the required media, the local copy will be served. | |
| mirrors | An associative array of available mirrors. Content will be copied to each of them | |

#### FTP Mirror configuration

The mirror configuration directives are:

| Directive | Description | Example |
|---|---|---|
| method | How the files are to be mirrored. Only FTP is currently supported. ftp (must be lowercase) | ftp |
| hostname | The hostname of the server to while the files will be mirrored. | ftp.mymirror.com |
| username | The username to use when logging into the mirror server. (Ask your server administrator if you are unsure) | user |
| password | The password to use when logging into the mirror server. (Ask your server administator if you are unsure) | password |
| directory | The directory into which to upload the mirrored media files. In general this should be an absolute path, but depending upon how your FTP login is setup it may be a relative path. (As your server administrator if you are unsure) | /usr/local/ftp/ca/images |
| passive | If set to '1' (the number one) then passive FTP connections are used. Passive connections are usually required if you are behind a firewall. | 1 |
| accessProtocol | The protocol to use in URLs that reference media on this mirror server. For simple web-served media like images this will usually be http or https. For streaming media this may be http, rtsp or rtmp. | http |
| accessHostname | The hostname to use in URLs that reference media on this mirror server. This is often, but not necessarily, the same as the hostname directive. | www.mymirror.com |
| accessUrlPath | The URL path (the part after the hostname) used to reference media on this mirror server. This is often, but not necessarily, the subset of the path set in the directory directive that is relative to a web server root. | /ca/images |

### 1.18.35 Monitor.conf

### 1.18.36 User_actions.conf

### 1.18.37 User_pref_defs.conf

## 1.19 Configuration File Syntax

A configuration file can contain any number of key-value pairs. Keys are simple alphanumeric text expressions. Values may be one of three types:

- **Scalar**: a string or number. Strings are always unquoted and may contain any character.
- **List**: a list of strings or numbers separated by commas and enclosed in square brackets ([ and ]). A string must be enclosed in double quotes if it contains a comma. You may not place the double quote character in a list item. Lists are retrievable as indexed PHP arrays. Lists may not be nested.
- **Associative array**: a list of key-value pairs. Both keys and values must be enclosed in double quotes if they contain commas. Neither may contain double quotes. Associative arrays are enclosed with curly brackets ({ and }). Separate keys from values with "=" Separate key-value pairs from each other using commas. Values may be strings, numbers or nested associative arrays. Associative arrays may be nested to any depth.
- **Keys** are always separated from values by "=" You may place as many spaces as you like on either side of the "=" character. Both lists and associative array may span as many lines as necessary.

Any line starting with a pound ("#") sign is considered a comment and ignored. It is ok to put leading spaces or tabs before a comment.

Note that if you begin a scalar value with a '[' or '{' character it will be parsed as a list or associative array respectively, which is not what you want. Be sure to precede the '[' or '{' with an exclamation point ('!') to indicate to the parser that you really want a scalar value.

An example configuration file, illustrating all three value types is below:

```
# 'email' is a simple scalar value
# all it does is set 'email' equal to the email address
#
email = support@collectiveaccess.org

# 'locales' is a list value
# The values in the list are available to CA as a simple '''ordered''' list.
#
locales = [en_US, de_DE, sp_AR]

# 'search_indexes' is an associative array (aka. a 'hash' in Perl-speak or a 'map' in␣
↪other languages)
# Each key in the array has associated with it a value, which can be either a simple␣
↪scalar (as in the case of the "sortable = yes" lines below) or a nested associative␣
↪array. You can nest arrays to any depth.
#
search_indexes = {
   objects = {
                 title = {
                          sortable = yes,
                          searchable = yes
                 },
                 description = {
                          sortable = no,
                          searchable = yes
                 }
   }
}
```

Substituting values Scalar values, once set, can subsequently be used, or substituted, in other locations in the configuration file. This allows you to set frequently used values at the beginning of a configuration file for use later in the file. If you need to change the value, you need only change it in one place - the place where it is defined.

Substituted values are simple scalar keys enclosed in "pointy" brackets (< and >), as in this example:

```
protocol = http
hostname = www.collectiveaccess.org

url = <protocol>://<hostname>
```

Here, we set the URL protocol (HTTP) and the hostname of the URL (www.collectiveaccess.org), then substitute the two values into the url key to create a valid URL.

### 1.19.1 Translating values into the user's current language

Any scalar values can be tagged for translation into the user's preferred language by enclosing them in a translation marker, like so: _("... my text ..."). Text so tagged with be passed to the GNU GetText translation function _t(), which is defined in *app/helpers/utilityHelpers.php*.

## 1.20 Introduction to Search & Browse Types

## 1.21 Search Syntax

No matter what back-end search engine you configure CA with, the Lucene search syntax is always used to specify queries. This helps to provide a consistent experience for users across implementations and also leverages the Lucene syntax, which is well designed and widely adopted. Note that not all back-end engines support all aspects of the Lucene syntax. In general you can rely upon core functionality always being supported: text searches, field-level limiting, parenthetical grouping and booleans. Features such as boosting, fuzzy matching and range searches may not be available in all engines.

### 1.21.1 Fulltext searches

To search across all indexed fields in the database (as defined in the search_indexing.conf configuration file) simply type in a word or words. Depending upon the engine, your input may be stemmed (suffixes removed before comparison with the index) to improve results. Most engines will only return results that contain all of the words specified, but some may employ logic to return seemingly relevant partial matches.

### 1.21.2 Limiting your search to a specific field

If you wish to restrict your search to a specific field in the CA database, specify the table name and field separated by a dot, like this:

`<table>.<field> (ex. ca_object_labels.name)`

A query in the form of

`ca_object_labels.name:Rollercoasters`

would return only objects whose label (eg. its title) contains the word rollercoasters

Note that this applies only to "intrinsic" fields that are hardcoded into the CA database. That is, they are always present no matter how you have CA configured - although you may not be using them. There are only a few of this type of field in common usage: label fields (for ca_objects, ca_entities, etc.), idno identifier fields (for ca_objects, ca_entities, etc.), and the extent and extent_units fields (for ca_objects and ca_object_lots).

### 1.21.3 Limiting your search to a specific metadata elements

Metadata elements are data fields specific to your installation. They may or may not exist in other installations. Searching on them is similar to searching on intrinsic fields:

`<ca_table>.<element code> (for example, ca_objects.description to search within the "description" element attached to objects.)`

You can see a list of all metadata elements (and their codes) available in your configuration in the metadata element editor available under the "System Configuration" option in the "Manage" menu. (Note: only system administrators have access to this editor).

In all engines you can perform text searches on any element. In some engines - specifically the MysqlFulltext engine that is the default engine upon installation - you can also perform specialized searches on certain types of elements. These searches are described in the following section.

### 1.21.4 Limiting searches to specific relationship types

By default, when searching on related content (Eg. search for objects using names of related entities) all relationships are considered. If you wish to limit your search to specific types of relationships append the relationship type code (or codes, separated by commas) following the field qualifier and a forward slash. For example this query:

```
ca_entities.preferred_labels.displayname/depicts:"Cynthia Hopkins")
```

when used to find objects will return all objects related to Cynthia Hopkins with a "depicts" relationship.

### 1.21.5 Searching on dates

To search on a date or date range, simply restrict your search to a date range element and then search on the desired date, using one of the formats described on the date and time format page. You can use any supported format and any precision - the search engine will find any date (and optionally times) that overlap your search date range. Matching is by default very loose: items with any overlap will be returned. You can restrict matching to items with dates that are completely encompassed by your search date by prepending a "#" to your search data. Eg. "#May 10 2005"

### 1.21.6 Searching on lengths and widths

To search on a length or width, restrict your search to a length or width element and use the desired quantity with units specified. You must specify units - there is no default no matter what your "units of measurement" preference is set to (this preference governs display of measurements only). If you want to find items that match a measurement exactly simply search on the quantity. CA will convert the quantity to the required units for comparison, so even if an item was measured in inches, a metric search will find it - if the measurements match of course.

```
ca_objects.width:12in
```

You can use almost any unit abbreviation listed on the measurement input format page. A few, such as " for inches and ' for feet have special meaning in the Lucene search syntax and should not be used.

If you want to search for items within a range of measurements, specify the upper and lower bounds of the range with units. The boundary values should be separated with the word "to" and enclosed in square brackets. Do not put spaces between the quantity and units. For example:

```
ca_objects.width:[12in to 24in]
```

would find all objects with a width between 12 and 24 inches (inclusive). Note that there is no space between "12" and "in" and "24" and "in"

### 1.21.7 Searching on numbers

Searching on numbers is very similar to searching on measurements, except that no units are necessary. To search on an integer or decimal value element restrict your search to the element and specify the number either singly or as a range. For example, to find objects with a user_ranking value of 5:

```
ca_objects.user_ranking:5
```

To find objects with user_ranking values between 1 and 5 (inclusive):

```
ca_objects.user_ranking:[1 to 5]
```

### 1.21.8 Searching on currency

Searching on currency is very similar to searching on numbers, except that a currency type is required. To search on an currency value element restrict your search to the element and specify the currency amount either singly or as a range. The amount should be prefixed with a three letter currency specified (eg. EUR for Euros, USD for US dollars) or one of the supports symbolic specifiers ($, ¥, £ and €). For example, to find objects with an appraisal_value value of $500:

```
ca_objects.appraisal_value:$500
```

To find objects with appraisal_value values $500 or under:

```
ca_objects.appraisal_value:[$0 to $500]
```

### 1.21.9 Searching on geographic locations

When searching on geographic locations, you have two options. You can either search within a bounding box specified by two latitude/longitude pairs or you can search for anything with a specified distance of a latitude/longitude point.

To search within a bounding box:

```
ca_objects.georeference:ca_objects.georeference:"[40.341,-71.011 to 45.322,
-75.963]"
```

Note that the latitudes and longitudes should be decimal and separated with “to”, ” - ” or “..”; the entire range should be enclosed in both square brackets (“[” and “]) and quotes. If you don’t use quotes on the part of the query up to the first space will be parts as geographic - not what you want.

To search the area within a specified radius of a point, use this kind of search:

```
ca_objects.georeference:ca_objects.georeference:"[40.5759250,-73.9911350 ~
5km]"
```

As with the bounding box query, enclose the search expression in square brackets and quotes. The maximum distance from the point can be specified in any of the units of length supported by the “Length” attribute type. The above query will find anything geocoded as being within 5 kilometers of the specified point.

### 1.21.10 Searching for blank values

As of version 1.4 you may search for item that have no content in a specific field using the special [BLANK] search term. [BLANK] must be used in conjunction with field specification and must be enclosed in double quotes. The following example will return all objects lacking descriptions:

```
ca_objects.description:"[BLANK]"
```

### 1.21.11 Access points

Typing ca_objects.description:grafitti every time you want to search for the word “grafitti” in the element “description” gets old quick, and certainly doesn’t look very pretty. To simplify the specification of field and element-limited searches, CA supports the definition of “access points.” Access points are simply lists of field and element specifications, defined in the search_indexing.conf configuration file, the names of which may be used in place of the actual specification. For example, you could do the ‘description’ search like this:

```
picText:grafitti
```

assuming that an access point like this was defined in search_indexing.conf:

```
picText = {
        fields = [ca_objects.description]
},
```

### 1.21.12 Boolean combination

Search expressions can be combined using the standard boolean "AND" and "OR" operators. Simply join together your search expressions with the words AND and OR. For example the query

`ca_objects.appraisal_value:[$0 to $500] AND ca_objects.description:broken`

will find all objects with BOTH an appraisal value of $500 or less and the word "broken" is their description. In contrast the query

`ca_objects.appraisal_value:[$0 to $500] OR ca_objects.description:broken`

will find objects with EITHER an appraisal value of $500 or less or the word "broken" in their description.

If you omit AND/OR between two search expressions, AND is assumed.

### 1.21.13 Wildcards

The asterisk ("*") is used as a wildcard character. That is, it matches any text. Wildcards may only be used at the end of a word, to match words that start your search term. For example:

`wri*`

would find records associated with words starting with the text "wri" Note that if your installation has "stemming" enabled, many English language words will automatically have their suffixes truncated and a wildcard appended. Thus, with stemming on, a query for "baking" or "baked" or "baker" would be transformed to "bak*" The stemmer is smart enough to not attempt truncation of a term you've added a wildcard to yourself. If you search for "bake*" the stemmer will leave it as-is.

### 1.21.14 Searching on creation and modification dates

You can search on the creation and modification dates of records using the special created and modified access points together with a valid date/time expression. For example, to find everything created on April 12, 2012 you can search using:

`created:"April 12 2012"`

or

`created:"4/12/2012"`

or with any other valid date/time expression. Any range will work, including ones that specify time and ones that are by month or year.

You can limit the returned items to those created or modified by a specific user by adding a valid user name after the access point. For example, to find things modified by user "catherine" on April 2012 you can search using:

`modified.catherine:"4/2012"`

Note that the user name is separated from the access point by a period ("."), and that the name of the user is their login user name, not their full name. Their login user name may be, but is not always, the user's email address.

### 1.21.15 Searching on counts

As of version 1.7 it is possible to index the number of relationships and repeating per metadata element for search. For relationships, counts may be broken out by relationship type, related item type, or both. Count queries are useful for locating records without specific relationships (eg. find objects without entities related as artist) or with potential problems (eg. find objects with between 10 and 100 related entities).

By default count indexing is only enabled on object-entity relationships, and broken out by relationship type. You may configure indexing of other counts in the search_indexing.conf configuration file.

Relationship counts may be queried using the relationship table name followed by the special count field. For example, in an object search to find all objects related to exactly one entities search for:

`ca_objects_x_entities.count:1`

To find all objects with exactly one entity related with the relationship type "artist":

`ca_objects_x_entities.count/artist:1`

To find all objects without related "artist" entities:

`ca_objects_x_entities.count/artist:0`

To find all objects with between 2 and 10 related entities:

`ca_objects_x_entities.count:[2 to 10]`

And to find all objects with between 2 and 10 related "artist" entities:

`ca_objects_x_entities.count/artist:[2 to 10]`

Note that the the table name used in these examples is "ca_objects_x_entities" rather than "ca_entities". When ca_objects_x_entities is indexed with count in search_indexing.conf (it is by default), counts are broken out by relationship type, which is what enables the count queries on relationship type.

You may also index counts on the related record itself (in this case ca_entities), breaking out counts by record type. Assuming your system is configured with the entity types "individual" and "organization" these queries would be possible:

Find objects with related organizations:

`ca_entities.count/organization:[1 to 100000]`

Find objects with only related individuals: <code>

`ca_entities.count/individual:[1 to 100000] and ca_entities.count/organization:0`

We use a range with an upper bound of 100000 here to ensure that we include objects with any number of entities. Expressions with < and > are not currently supported.

Similarly, the number of values present for each metadata element is indexed and may be queried. This can be useful for locating records that lack a value in a field, or have many values. For example:

To find all object records that lack at least one value in the "dimensions" field:

`ca_objects.dimensions.count:0`

To find all objects that have more than 5 values in the "dimensions" field:

`ca_objects.dimensions.count:[5 to 100000]`

As with any other search field specification, you may create more convenient aliases for commonly used counts in search_indexing.conf by creating an access point.

## 1.22 Indexing Options

## 1.23 Search Engines

### 1.23.1 SQL Search

SQLSearch is an engine that employs regular MySQL tables to create an inverted index stored. This technique was used in the 0.5x version of CollectiveAccess and provided reasonable performance and scalability combined with easy deployment (zero-configuration is required). For version 0.6 and 1.0 alternative engines were explored that leveraged existing code (eg. PHP Lucene, MySQL FULLTEXT, SOLR, etc.). While ultimately workable, none of the other options combine the deployment and performance characteristics of the inverted index approach. Thus, a new Unicode-friendly "SQL Search" plugin has been implemented, as of version 1.1, as an alternative to PHP Lucene and MySQL Fulltext, the other "easy deploy" options. As of version 1.1, SqlSearch is the default search engine option and as of version 1.3 the only supported "easy deploy" option.

Pros: Performance and scalability are generally good; deployment is effortless

Cons: Indexing can be slow; disk space requirements for indices can be large

Status: Implemented

### 1.23.2 ElasticSearch

ElasticSearch is a simple, fast and increasingly popular search engine.

Pros: Performance and scalability are very good

Cons: Requires you to run an ElasticSearch installation, which means running a Java application stack. This is often not an option for installations with limited IT resources.

About ElasticSearch ElasticSearch is an an Open Source (Apache 2), Distributed, RESTful, Search Engine built on top of Apache Lucene. CollectiveAccess added support in v1.3 and completely overhauled the plugin again in v1.6. It now uses the official ElasticSearch PHP API and no longer requires running a script to maintain the field mapping for the search engine.

Note that CollectiveAccess v1.6 only supports ElasticSearch 2.0 or higher!

Setup Please refer to the ElasticSearch website for installation and setup notes. Prebuilt packages are available for apt/dpkg on Debian/Ubuntu, yum on CentOS/RedHat or homebrew on Mac OS X, and probably your favorite package manager too!

Once you have ElasticSearch set up, you will have to set aside an index for CollectiveAccess to use. By default that index is called "collectiveaccess" but that can be changed in the search.conf configuration file if you want to use one ElasticSearch setup for multiple CollectiveAccess instances. You also have to configure the communication endpoint you want CollectiveAccess to use. If you're running ElasticSearch locally with the default settings, the default values in the config file should work as is: localhost and port 9200.

search_elasticsearch_base_url = http://localhost:9200/ search_elasticsearch_index_name = collectiveaccess Interaction with the CollectiveAccess Providence Installer CollectiveAccess will try to maintain the mapping automatically and it uses a few local caches to do so. When running the installer, these cache interactions can get a little wonky and lead to seemingly random exceptions and errors. Try to make sure you delete your ElasticSearch index and clear all local caches (by deleting the contents of app/tmp) before you run the installer if you have CollectiveAccess configured to use ElasticSearch before you install.

You could also run the installer using SqlSearch and then switch to ElasticSearch later. Remember to reindex your database contents after you switch.

Operation The v1.6 implementation of the ElasticSearch Plugin should take care of mapping maintenance automatically. If you see mapping exceptions piling up in the ElasticSearch log, that's probably because you've changed things in search_indexing.conf.

One common problem is having different analyzer/tokenizer settings for the same field in different contexts. For instance, you could have ca_object_lots.idno_stub indexed for ca_objects with no additional settings but then use INDEX_AS_IDNO and BOOST when indexing it for ca_object_lots. ElasticSearch doesn't like that. Try streamlining these settings so that they're the same for every occurrence of a field in search_indexing.conf. The default config that ships with Providence should be fine but if you have local changes, keep an eye on the ElasticSearch log and change your indexing config accordingly.

## 1.24 Introduction to Media Management

- *Supported media formats*
- Media attribute
- Media representation bundle
- *Media volumes*
- *Media mirroring*
- *Media processing*
- Media watermarking
- Media display
- Media metadata
- Media embedded metadata
- Media integrity
- Media viewer
- Media import
- Reprocessing media
- Media background processing

## 1.25 Media Mirroring

### 1.25.1 Mirroring media uploaded to CollectiveAccess via FTP

All media uploaded to CollectiveAccess are stored locally in web-accessible locations defined in the media_volumes.conf configuration file. This is usually all you need, but there are a few cases where being able to automatically copy – or mirror - media to another server can come in handy:

You want to manage audio or video files in CollectiveAccess locally but stream them from a dedicated audio/video host externally, to save local bandwidth or take advantage of features only streaming hosts can provide.

You want to provide high-availability to media files using multiple servers.

You want to make real-time backups of your media.

CollectiveAccess supports mirroring via FTP ("File Transfer Protocol"). FTP is a standard for moving files between servers on the Internet. When FTP mirroring is enabled, every time you upload a new media file the file and its derivatives will be automatically copied using FTP to one or more remote servers. The copying happens in the background so you need not wait for the file transfers to complete before continuing your work. Files of any size (subject to disk space limitations of the mirror servers, of course) may be mirrored.

**Note:** The media mirroring system is very old. It was actually part of the 0.5x code base and was originally intended to move files to Adobe Flash Streaming Server.

At the time everyone still used FTP to move files.

Additionally, CA can be configured to automatically use a mirror to serve media internally, if one is available. In this case when a media file is first uploaded it will be served from the local location. Once mirroring is complete (for large files or limited bandwidth connections, this can take a while) CA will automatically shift over to using the mirror.

Note that the CA mirroring system will automatically remove media from mirrors if they are removed from the local system. This prevents waste of disk space on mirror servers. You should not rely upon mirrors for backups of accidentally deleted files. Use a traditional data backup solution instead.

### Configuring mirroring

Mirroring is configured on a per-volume basis. All media written to a mirrored volume are mirrored. If you want to mirror a specific type of file then you should create a separate volume for that type of file and configure the media processing system (in media_processing.conf) to use the volume. The default CA configuration has definitions for media-specific volumes that should work fine for most installations.

Within a given volume, to enable mirroring add a "mirrors" list directive. The mirrors list should contain a series of "mirror codes" (unique alphanumeric identifiers for the mirror that you define), each of which has a list of configuration values for the mirror. A typical mirror setup to mirror all media added to the "images" volume (taken from the CA default configuration) looks like this:

```
images = {
    hostname = <site_hostname>,
    protocol = <site_protocol>,
    urlPath = <ca_media_url_root>/images,
    absolutePath = <ca_base_dir>/<ca_media_url_root>/images,
    writeable = 1,
    description = Images,

    accessUsingMirror = my_mirror,

    mirrors = {
        my_mirror = {
            method = ftp,
            hostname = ftp.mysite.org,
            username = my_ftp_login_name,
            password = my_password,
            directory = /usr/home/mysite/public_html/images,
            passive = 1,
            accessProtocol = http,
            accessHostname = www.mysite.org,
            accessUrlPath = /images
        }
    }
}
```

Note the `accessUsingMirror` directive. This tells CA what mirror to use locally if it is available. If you omit this directive the mirror will receive files from CA but will not actually be used by CA to serve media.

For further details about the mirror configuration see *FTP Mirror configuration*.

## 1.26 Display Template Syntax

Display templates are used to format data from bundles (elements of metadata stored in CollectiveAccess) for display on screen, output into reports and presentation in search results. When no display template is defined CollectiveAccess defaults to displaying bundle data in the simplest possible way, typically as a semicolon-delimited list of values. For bundles comprised of a single value (Eg. a simple text metadata element) this is often enough. For complex bundles consisting of several discrete values – a mailing address for example – a template is usually required to adequately format the value. Other cases where bundle display templates are called for:

- To define styling, such as headings, bold and italics around bundle elements.
- To format and conditionally include delimiters and suffixes between values in a complex bundle. For example, in a bundle with width, length and height dimensions, "x" delimiters can be placed between each dimension value. A display can be used to output values in a width/length/height order (or any other order). Thus a bundle with length=24", height=8" and width=3" can be output as 3" x 24" x 8" . . . or 3"W x 24"L x 8"H . . . or 3"W x 8"H if length happens to be undefined (because displays can intelligently omit the delimiter and suffix).
- To display data attached to related items. For example to display both the name and life dates for related entities a bundle display template can be used to extract and format the data. Any data attached to the related entity can be displayed.
- To display related data traversing any number of intervening relationships. As a simple example, imagine that you have an object related to a collection, and the collection is related to a donor. It's not necessary to catalog the donor directly on the object in order to display the donor's address there, because it's possible to pull the address through the intervening collection relationship. Another example prevalent in film and performance archives, is that objects can be related to "works" (occurrences) which in turn have entity relationships ("director", "actor",

"choreographer"). A display template can display object information alongside entity data related to a work that is related to the object.

- To apply one of several display formats using expressions conditional on one or more data values.

Display templates are also used extensively by Pawtucket 2.0 for formatting in themes. They are the preferred formatting method in Pawtucket 2.0, although mixed HTML/PHP coding is still supported.

### 1.26.1 Defining templates

Default display templates can be defined for metadata elements as part of their configuration. Default formatting can be overridden by additional context-specific templates within a display or user interface.

The default template for a metadata element can be set in the configuration interface. Display and user interface related templates may be set in their respective configuration editors on a per-bundle basis. When a template is defined for a metadata element within a display or editor user interface it will take precedence over templates defined in the element's configuration.

### 1.26.2 Template syntax

At their most basic, templates are simply text with placeholders to be replaced by bundle values. Placeholders always start with a caret ("^") character followed by a bundle specifier, an unambiguous identifier for a metadata element. For example, if you have a metadata element in an object record with the element code description and wish to preface the value of the element with the label "Description:" the template would be:

```
Description:  ^ca_objects.description
```

where ca_objects indicates an object record and description is the metadata element code.

If the value of the description metadata element happens to be empty, this template will cause the label "Description:" to be awkwardly displayed without a trailing value. To avoid unwanted blank spaces a display template can be made conditional on the presence of a value within a field. A template for description that only displays something if there's a description available would look like this:

```
<ifdef code="ca_objects.description">Description:  ^ca_objects.description</
ifdef>
```

Everything between the <ifdef> and </ifdef> is only output for the corresponding bundle (specified without the ^ in the <ifdef> tag because it's a code, not a placeholder in this context) when it actually has a value.

Conditional output can be used for more than just labels. For dimensions and other collections of quantities, conditional output can be used to deal with variations when not all values are available in all cases. For example, let's say you have a metadata container on an object record named "dimensions" with three sub-elements: width, height and depth, all of which are elements of type Length. Displaying the container ca_objects.dimensions without a template would result in three values separated with semicolons, which are the default delimiter:

```
12"; 6"; 9"
```

(we assume here that we're displaying in English units)

To make it clearer we can format the container using this template:

```
^ca_objects.dimensions.width W x ^ca_objects.dimensions.height H x
^ca_objects.dimensions.depth D
```

This will display:

```
12" W x 6" H x 9" D
```

As you can see, a special syntax is used to articulate container elements. It is no longer just ^ca_objects.dimensions in our example, but rather the code for the parent container along with the specific sub-element you've chosen to display.

If the depth value happens to be blank in some cases then the output would sometimes be like this:

```
12" W x 6" H x D
```

To rectify this we can use conditional output:

```
<ifdef code="ca_objects.dimensions.width">^ca_objects.dimensions.width W x</ifdef>
↪<ifdef code="ca_objects.dimensions.height">
^ca_objects.dimensions.height H x</ifdef> <ifdef code="ca_objects.dimensions.depth">^
↪ca_objects.dimensions.depth D</ifdef>
```

Note that we can also use conditionals to close up the space between ^ca_objects.dimensions.width and the "W", ^ca_objects.dimensions.height and "H" and ^ca_objects.dimensions.depth and "D". Normally space is required between the placeholder and any non-placeholder text to make clear where the placeholder ends. With a conditional you can keep the placeholder separate from other text without resorting to spaces, as in this example:

```
^ca_objects.dimensions.width<ifdef code="ca_objects.dimensions.width">W x</ifdef> ^ca_
↪objects.dimensions.height
<ifdef code="ca_objects.dimensions.height">H x</ifdef> ^ca_objects.dimensions.depth
↪<ifdef code="ca_objects.dimensions.depth">D</ifdef>
```

If you need to make part of your template conditional upon more than one value being set simply list the placeholder names in the "code" value separated by commas:

```
<ifdef code="ca_objects.dimensions.width,ca_objects.dimensions.height,ca_objects.
↪dimensions.depth">Dimensions are: </ifdef>
^ca_objects.dimensions.width<ifdef code="ca_objects.dimensions.width">W
x</ifde> ^ca_objects.dimensions.height<ifdef code="ca_objects.dimensions.height">
H x</ifdef> ^ca_objects.dimensions.depth<ifdef code="ca_objects.dimensions.depth">D</
↪ifdef>
```

"Dimensions are:" will only be output if width, height and depth all have values. The text can be output if any of the values in the code list are set by separating the placeholder names with "|" (aka. "pipe") characters:

```
<ifdef code="ca_objects.dimensions.width|ca_objects.dimensions.height|ca_objects.
↪dimensions.depth">Dimensions are: </ifdef>
^ca_objects.dimensions.width<ifdef code="ca_objects.dimensions.width">W x</ifdef>
^ca_objects.dimensions.height<ifdef code="ca_objects.dimensions.height">H x</ifdef>
^ca_objects.dimensions.depth<ifdef code="ca_objects.dimensions.depth">D</ifdef>
```

There are some cases in which you may need to make part of a template conditional upon a value or values not being defined. The <ifnotdef> tag will do this in an analogous manner to <ifdef>. For example, if you want to output a "No dimensions" message when no values are defined:

```
<ifnotdef code="ca_objects.dimensions.width,ca_objects.dimensions.height,ca_objects.
↪dimensions.depth">No dimensions are set</ifnotdef>
^ca_objects.dimensions.width<ifdef code="ca_objects.dimensions.width">W x</ifdef> ^ca_
↪objects.dimensions.height
<ifdef code="ca_objects.dimensions.height">H x</ifdef> ^ca_objects.dimensions.depth
↪<ifdef code="ca_objects.dimensions.depth">D</ifdef>
```

### 1.26.3 Placeholder options

Placeholder values may be modified by options appended as a series of named parameters. Options are separated from the placeholder with a "%" character and listed in <name>=<value> pairs delimited by "&" or "%" characters.(("&" are used in older templates, but now may be used interchangeably with "%"). For example:

```
^ca_objects.hierarchy.preferred_labels.name%maxLevelsFromBottom=4&delimiter=__
```

will output a list of hierarchical object titles consisting of the bottom-most four titles separated by arrows. If those options were not set they would revert to defaults, in this case the entire hierarchy delimited by semicolons.

Any number of options may be appended to a placeholder.

Note that spaces are not allowed in options as they are used to separate placeholders. You can use URL encoding (eg. %20 for a space) or a underscores in place of spaces.

The following options may be used to format the text value of any placeholder:

| Placeholder | Description |
|---|---|
| toUpper | Forces text to all upper case when set to a non-zero value. |
| toLower | Forces text to all lower case when set to a non-zero value. |
| makeFirstUpper | Make first character of text upper case when set to a non-zero value. |
| useSingular | Converts label to singular when set to a non-zero value. |
| trim | Trim white space from beginning and end of value. |
| start | Trim the beginning of the text such that it begins at the specified character; ex. "This is a test" with start=2 will be transformed to "is is a test" |
| length | Truncate the text to the specified number of characters. Can be used with the start option to extract sections of text, or alone to ensure text does not go beyond a maximum length. |
| truncate | Truncates the text to the specified maximum length. The equivalent of setting the start option to zero and length option to the truncation length. |
| ellipsis | Add an ellipsis ("...") to truncated text when set to a non-zero value. The resulting text will be the specified length including the three character ellipsis. That is, text truncated to 12 characters will include 9 characters of text and 3 characters of ellipsis. |

For simple true/false options such as toUpper you may omit the "=" and value. These two templates are the same:

```
^ca_objects.preferred_labels.name%trim=1
```

and

```
^ca_objects.preferred_labels.name%trim
```

### 1.26.4 Pulling metadata through a relationship

In the previous examples, data displayed is always from a particular object record at hand – the "primary" record. Templates are always processed relative to to the primary record. If you are formatting object search results, for example, your template will be repeatedly evaluated for each object in the result set, with each object taking its turn as primary. It's obvious but still worth stating: placeholders referring directly to data in the primary (^ca_objects.idno for example) derive their values from the primary. If a bundle repeats for a record, you may get multiple values, but all values referring to the primary will always be taken from the primary. Any record can be primary. Primary-ness is simply the context is which a template is processed.

It is often necessary to display metadata from records related to the primary. For example, you might want to display entities related to an object (the primary) displaying each entity's lifespan and birthplace next to their name. Or a display the related collections, with name, access restrictions and availability information. Or perhaps a display of objects related to the current primary object.

For simple cases displaying related data is similar to primary data. For placeholders that refer to non-primary data CollectiveAccess will look for records of that kind directly related to the primary. For a ^ca_entities.preferred_labels.displayname placeholder in a display for object results, CollectiveAccess will pull the names of all entities directly related to the primary object. Using our sample data:

`^ca_entities.preferred_labels.displayname`

will result in a list of display names for related entities, separated by semicolons (the default delimiter):

`George Tilyou; Elmer Dundy`

To pull data from related records of the same kind as the primary (Ex. objects related to an object) add "related" to the bundle specifier:

`^ca_objects.related.preferred_labels.displayname`

With our sample data this will result in the title of the object related to the primary being returned. You can include "related" in specifiers for any kind of related record but it is only required when things would otherwise be ambiguous without it.

You may pull any data in the related entity records using similarly constructed placeholders. For example, this template:

`^ca_entities.preferred_labels.displayname (Life dates: ^ca_entities.life_span)`

will return

`George Tilyou; Elmer Dundy; (Life dates: 1865 - 1914; 1862 - 1907)`

Each placeholder is evaluated separately and a list of values returned in its place. To format several related data elements in a block, as well as to display indirectly related data (such as the related entity's birthplaces), set custom delimiters and other options a new template directive, the <unit> tag, is needed.

### 1.26.5 Formatting templates with <unit>

<unit> tags allow you to break your templates into sub-templates that are evaluated independently and then reassembled for final output. Using the <unit> relativeTo attribute, the primary record of the template may be transformed into one or more related records, repeating values from the primary (e.g. values in a repeating container) or a set of hierarchical values, and the sub-template evaluated for each.

<unit>'s and relativeTo enable a host of useful (and often complex) formatting transformations:

- When a record has repeating containers. Say you have a repeating address container on an entity record to accommodate multiple address changes. If you format your display template without specifying that each instance of the container needs to be displayed as a unit the result will be a single address in return, no matter how many addresses are entered, and each placeholder will contain the values for all of the addresses - a nonsensical way to display an address list. Wrapping the address portion of the template in <unit> tags and specifying that it be evaluated relative to the repeating address element, rather than the primary record itself, will force the template contained within to be evaluated once per repeating address value, resulting in an independently formatted value for each address. Ex.

```
<unit relativeTo="ca_entities.address">
^ca_entities.address.street_address<br/>^ca_entities.address.city, ^ca_entities.
↪address.state ^ca_entities.address.zip_code<br/>
</unit>
```

The relativeTo option in the <unit> tag forces the sub-template to be evaluated once per address value in the primary record.

- When you need to present more than one data element from related records side-by-side. In the previous section we saw how different placeholders referencing the same related records always return separate lists, one per placeholder. When displayed side-by-side the result is a series of lists rather than the discrete blocks of output for each related item that are more typically desired. <unit> tags make it possible to define sub-templates that are evaluated repeatedly, as many times as there are related records. Our example in the previous section reformatted with <unit> tags like this:

```
<unit relativeTo="ca_entities">^ca_entities.preferred_labels.displayname (Life
dates:  ^ca_entities.life_span)</unit>
```

results in this output:

```
George Tilyou (Life dates:  1865 - 1914); Elmer Dundy (Life dates:  1862 -
1907)
```

Here the relativeTo option in the <unit> tag shifts the primary record to be each related entity in turn, in the sub-template defined by the <unit> only.

- When you need to set display options for part of a template. <unit> tags provide options to modify output for sub-templates. You can set the delimiter for repeating values using the delimiter option, or restrict the related items displayed by relationship type or related item type using restrictToRelationshipTypes and restrictToTypes respectively (or their counterparts excludeRelationshipTypes and excludeTypes). (You can also set options on individual placeholders, but declaring options on <unit> tags is usually more convenient and always more readable). Ex.

```
<unit relativeTo="ca_entities" restrictToRelationshipTypes="actor, director, producer
↪">
^ca_entities.preferred_labels.displayname (Life dates: ^ca_entities.life_span)
</unit>
```

- When you need to display metadata relating to hierarchical records. Without the <unit> tag, there's no way to individually list child records and accompanying metadata in a display. With the <unit> tag you can display parent and/or child records and hierarchical paths as discrete, complex units, by making the unit relativeTo the hierarchical record set. Ex.

```
<unit relativeTo="ca_list_items.hierarchy"><p>^ca_list_items.preferred_labels.
name_plural (ca_list_items.idno)</p></unit>
```

Here the relativeTo option in the <unit> tag shifts the primary record to be each related list item in the hierarchy in turn, in the sub-template defined by the <unit> only.

- When you need to pull metadata through an indirect relationship. Without the <unit> tag only metadata from records directly related to the primary can be displayed in a template. In our sample data, this means only the entities related to the primary object can be displayed. The birthplace data related to each entity cannot. By using <unit> tags nested within one another and specifying the relativeTo option we can shift the primary record for a sub-template across any number of relationships. We might call this "Six Degrees of Kevin Bacon for CollectiveAccess" where A is related to B which is related to C. For example, if the primary is an object, and you need to display place data from entities related to objects (not places related directly to the object), the following template would do the job:

```
Object is ^ca_objects.preferred_labels.name;
Entities are: <unit relativeTo="ca_entities">^ca_entities.preferred_labels.displayname
(Birthplace: <unit relativeTo="ca_places">^ca_places.preferred_labels.name</unit></
↪unit>
```

Each unit shifts the primary by one relational "jump." Nesting <units> allows shifts to accumulate because they are always evaluated relative to their context. Thus entities related to objects are grabbed, and then places related to those entities.

<unit> tags may take any of the following attributes:

| Attribute | Description | Default | Supported |
|---|---|---|---|
| relativeTo | Transforms the primary record of the template or enclosing &lt;unit&gt; (when &lt;unit&gt;'s are nested) to a set of related records, hierarchically related records or repeating values. | None; must be set | |
| restrictToTypes | For &lt;unit&gt;'s relativeTo a relationship (eg. relativeTo='ca_entities') or hierarchy (eg. relativeTo='ca_objects.hierarchy', relativeTo='ca_objects.parent', relativeTo='ca_objects.siblings', relativeTo='ca_objects.children') will restrict the record set to those of the specified types. Use type identifiers and list multiple types separated by commas. | None – no restriction | |
| restrictToRelationshipTypes | For &lt;unit&gt;'s relativeTo a relationship (eg. relativeTo='ca_entities') will restrict the record set to those related with the specified relationship types. Use relationship type codes and list multiple codes separated by commas. | None – no restriction | |
| excludeTypes | For &lt;unit&gt;'s relativeTo a relationship (eg. relativeTo='ca_entities') or hierarchy (eg. relativeTo='ca_objects.hierarchy', relativeTo='ca_objects.parent', relativeTo='ca_objects.siblings', relativeTo='ca_objects.children') will restrict the record set to those NOT of the specified types. Use type identifiers and list multiple types separated by commas. | None – no restriction | |
| excludeRelationshipTypes | For &lt;unit&gt;'s relativeTo a relationship (eg. relativeTo='ca_entities') will restrict the record set to those related with relationship types NOT in the list. Use relationship type | None – no restriction | |

The <unit> tag presents many opportunities for complex display formatting which are explained in more detail, along with examples, here.

You can limit the number of values returned from a <unit> operating on a repeating value using the start and limit unit attributes described previously. You can display text indicating how many values were not shown using the <whenunitomits> tag following a <unit>. For example, to show the first 5 related entities and then a message with the total number:

```
<code>
<unit relativeTo="ca_entities" delimiter=", " start="0" length="5">^ca_entities.
↪preferred_labels.displayname</unit><whenunitomits> and ^omitcount more</
↪whenunitomits>
</code>
```

The ^omitcount placeholder can be used within the <unit> or <whenunitomits> tag. The <whenunitomits> tag always refers to the number of values omitted in the <unit> before it in the template and will be suppressed when no values from the previous <unit> are hidden.

### 1.26.6 Contextual tags: <more> and <between>

Templates using <ifdef> and <ifnotdef> can get long and unruly when they include many elements dependent on the state of multiple placeholders. To help make such templates more manageable two tags are available that control output based solely upon their position in a template, obviating the need for long lists of placeholder names.

The <more> tag will output content if any placeholders following it have a value. Thus this template:

`^ca_objects.description <more><br/>The source for this was:  </more>^ca_objects.description_source`

will output this (assuming both description and description_source are set to “A metal pan” and “1978 auction catalogue” respectively):

```
A metal pan
The source for this was: 1978 auction catalogue
```

If description_source was empty the output would be:

`A metal pan`

The <between> tag will output content if any placeholders before it in the template and the placeholder directly following it in the template have values. This makes delimiting lists of values more compact than options using <ifdef>:

`^ca_objects.dimensions.width <between>x</between> ^ca_objects.dimensions.height <between>x</between> ^depth`

The output of this would be the defined dimensions with a single “x” delimiter between each pair.

### 1.26.7 Conditional tags: <ifdef>, <ifnotdef>, <ifcount>, <if>

As mentioned earlier you can make display of portions of your template contingent upon specified conditions by surrounding part of the template with <ifdef> and <ifnotdef> tags. Both tags take a “code” attribute containing one or more bundle specifiers. If the value for the bundle is not empty <ifdef> will display the portion of the template it encloses. Conversely, if the value is empty <ifnotdef> will display the content it encloses.

For example:

`Title:  ^ca_objects.preferred_labels.name <ifdef code="ca_objects.description">Description:  ^ca_objects.description</ifdef>`

Note that the specifier in the code attribute is not a placeholder and therefore does not take a "^" prefix.

You can make <ifdef> and <ifnotdef> contingent upon more than one bundle by listing them in the code attribute separated by commas or pipes ("|"). When separated by commas, all of the bundles must be defined (<ifdef>) or not defined (<ifnotdef>) for the tag to display content. When separated by pipes, any of the bundles defined (<ifdef>) or not defined (<ifnotdef>) will cause the tag to display content.

The <ifcount> tag controls display of content based upon the number of values available from the bundle specifier in code. It is useful when you wish to only show content when the number of values a bundle has is within a range. For example, if you wish to show a list of related entities only when there are between 2 and 5 relationships:

`<ifcount code="ca_entities.related" min="2" max="5">Related entities: ^ca_entities.preferred_labels.displayname</ifcount>`

You can show content whenever the count is greater than a number by omitting the max attribute:

`<ifcount code="ca_entities.related" min="2">Related entities:  ^ca_entities.preferred_labels.displayname</ifcount>`

If the min attribute is omitted it is assumed to be zero.

To only show content when the count is a specific number set both min and max to the same number:

`<ifcount code="ca_entities.related" min="1" max="1">Related entity: ^ca_entities.preferred_labels.displayname</ifcount>`

The <if> tag provides maximum control by using *expressions* to determine when content is displayed. For example, to output the display only if "current" is selected from the type drop-down in a repeating credit line container:

```
<unit relativeTo="ca_objects.credit_line"><if rule=\"^credit_type =~ /current/\">^ca_
↪objects.credit_line.credit_text
(^ca_objects.credit_line.credit_type)</if></unit>
```

The rule attribute must be set to a valid expression, which can use any valid placeholder available in the template, and must be enclosed in escaped (prepended "") quotes to ensure that it is evaluated correctly.

Both <ifcount> and <ifdef> include blank values in their evaluation. From version 1.7.9 blank values may suppressed by setting the optional "omitBlanks" to a non-zero value. This is often useful when formatting data for display. If "omitBlanks" is set, <ifcount> will return the number of non-blank values; <ifdef> will evaluate as true only if the bundle has at least one non-blank value. Note that <if> does not support the "omitBlanks" option. You must filter blank values in the expression.

### 1.26.8 Even more conditional: the <case> tag

Sometimes you need to to choose from one of several templates based upon varying criteria. For instance, when listing entities related to an object you might want to vary the text before the list with respect to the number of entities being listed. There are ways to do this with display templates, but the cleanest way is with a <case> tag:

```
<case>
        <ifcount code="ca_entities.related" max="0">No related entities</ifcount>
        <ifcount code="ca_entities.related" min="1" max="1">Related entity: ^ca_
↪entities.preferred_labels.name</ifcount>
        <ifcount code="ca_entities.related" min="2">Related entities: ^ca_entities.
↪preferred_labels.name%delimiter=,_</ifcount>
</case>
```

The <case> tag evaluates each <ifcount> tag in order and stops at the first one that results in output. You can include templates beginning with <ifdef>, <ifnotdef> and <if> as well as <ifcount>. If a <unit> tag is included as the last template in a <case> it will be used as the default in case no other template results in output.

Because <case> tags stop evaluating as soon as they find a template with output they are generally the best performing way to choose a template from a list of possibilities.

### 1.26.9 Expressions

It's also possible to output the result of *expressions* as-is. A use case for this is making certain statistics about your metadata searchable. For instance, you could use Prepopulate to always keep the current number of entity relationships for your objects in a hidden (but searchable and sortable) field.

Usage of the expression tag is simple: Anything inside the tag is treated as an *expression*. You can use your typical caret-prefixed bundle placeholders and even unit tags. Unit tags get evaluated/replaced first when CollectiveAccess runs display templates, so you can use the result of a unit tag in your expression. Here are a few basic examples:

`<expression>5 + 4</expression>` `<expression>length(^ca_objects.preferred_labels)</expression>`

This one outputs related entity names and their string lengths:

`<unit relativeTo="ca_entities">^ca_entities.preferred_labels, <expression>length(^ca_entities.preferred_labels)</expression></unit>`

The following counts the number of entity relationships for the current record. We use a unit tag to generate the parameters for the sizeof function.

`<expression>sizeof(<unit relativeTo="ca_entities" delimiter=",">^ca_entities.entity_id</unit>)</expression>`

This one calculates the age of Alan Turing:

`<expression>age("23 June 1912", "7 June 1954")</expression>`

### 1.26.10 Formatting hierarchical displays

Many types of records can be arranged hierarchically. To get some or all of the hierarchy for display use a hierarchical bundle specifier. This is just a normal specifier with a hierarchical modifier (hierarchy, parent, children) added.

For example, for an object primary, a ^ca_objects.hierarchy.preferred_labels.name placeholder will return the names of all objects in the hierarchy from top to bottom. You'll probably want to set a delimiter between each item in the hierarchy. You can do so by adding a placeholder option: append a percent sign and delimiter=<my delimiter> to the bundle specifier, like so:

`^ca_objects.hierarchy.preferred_labels.name%delimiter=__`

When setting the delimiter, underscores are used in place of spaces. Spaces are used to delimit individual bundle specifiers, so you can't have the delimiter floating out past a space associated with the specifier. The underscores will be converted to spaces for display.

You can get more control over hierarchy displays using a <unit> set relative to a hierarchy. For our object primary:

`<unit relativeTo="ca_objects.hierarchy">^ca_objects.preferred_labels.name (^ca_objects.idno)</unit>`

will evaluate the <unit> for each record in the hierarchy in turn set to primary. Related data can be accessed as well, and additional <unit>'s can be specified within.

The parent and children modifiers work similarly to hierarchy but return the immediate parent of a record or its immediate children respectively.

There are a number of placeholder options that can be used to modify how hierarchical data is displayed:

| Option | Description | Default |
|---|---|---|
| delimiter | Text to use as delimiter for multiple values. | ; |
| maxLevelsFromTop | Restrict the number of levels returned to the top-most beginning with the root. | None |
| maxLevelsFromBottom | Restrict the number of levels returned to the top-most beginning with the root. | None |
| hierarchyDirection | Order in which to return hierarchical levels. Set to either "asc" or "desc". "asc"ending returns hierarchy beginning with the root; "desc"ending begins with the child furthest from the root. | asc |
| allDescendants | Return all items from the full depth of the hierarchy when fetching children using the children modifier. By default only immediate children are returned. | FALSE |

## 1.26.11 Making links to other records

The <l> tag may be used to create links within the template. The links will always point to the primary record. In Providence the link will lead to the editing interface for the record; in Pawtucket the link will be to the detail display for the record. It is possible to write plugins that override this behavior and create other sorts of links.

Any stretch of the template may be made into a link. For example, assuming the primary is an entity:

```
<l>^ca_entities.preferred_labels.displayname</l> <ifdef code="ca_entities.address.
↪address1">(</ifdef>^ca_entities.address.address1
<ifdef code="ca_entities.address.address1">)</ifdef>
```

Clicking on the entity name in Providence would take a cataloguer to the editor for the entity record; in Providence it leads to the detail for the entity.

Links always point to the primary record. If you use <l> tags within a <unit> the links will be to the primary within the <unit>.

## 1.26.12 Using HTML

You can freely use HTML tags for formatting within your templates, so long you follow the rules and use well-formed markup. Be sure to close any tag you open. The special template tags such as <ifdef> count in terms of well-formedness even though they don't display. This, for instance, is not correct and will render unpredictably:

```
<l>^ca_occurrences.preferred_labels.names</l> <ifdef code="ca_occurrences.exhibit_date
↪"><b>(Dates: </ifdef>^ca_occurrences.exhibit_date
<ifdef code="ca_occurrences.exhibit_date">)</b></ifdef> ^ca_occurrences.description
```

Notice that the <b> tag in the first <ifdef> is not closed before the closing </ifdef>, producing invalid markup. There is a </b> tag later on but this too is taken on its own due to the enclosing <ifdef> tags. The correct way to write this template is:

```
<l>^ca_occurrences.preferred_labels.names</l> <ifdef code="ca_occurrences.exhibit_date
↪"><b>(Dates: ^ca_occurrences.exhibit_date
</b></ifdef> ^ca_occurrences.description
```

### 1.26.13 Special placeholders

There are a few placeholders that have special meanings for certain kinds of primary records:

| Placeholder | Description |
|---|---|
| ^date | Displays the current date. Valid in any template regardless of the kind of primary. Date is formatted as "day month year" (ex. "10 January 2014") unless a format is specified using the format placeholder option. The option takes a PHP date()-style formatting string. See [1]. (Ex. ^date%format=c) |
| ^relationship_typename | Displays the name of the relationship type when the primary is a relationship record such as ca_objects_x_entities. Note that templates pulling related records in bundle displays are evaluated relative to the primary representing the relationship, not the related record. Thus this and the other ^relationship_* placeholders are available by default when pulling related data in bundle displays |
| ^relationship_type_id | The internal numeric type_id for the relationship type when the primary is a relationship record such as ca_objects_x_entities. |
| ^relationship_typecode | The alphanumeric code for the relationship type when the primary is a relationship record such as ca_objects_x_entities. |
| ^primary | Displays the name of the primary for the template or current <unit> sub-template. This can be useful for debugging. |
| ^count | Displays the number of values in the current primary for the template or current <unit> sub-template. |
| ^index | Displays the one-based index of the current value in the primary or <unit> sub-template. As a <unit> iterates through each value ^index will increment by one until it reaches ^count. |

As of version 1.7.9 there are also several special placeholders available for object representations that return pre-formatted media-specific metadata. These are typically used to format display text in lists of object representations:

| Placeholder | Description | Options | Examples |
|---|---|---|---|
| ^ca_object_representations.transcription_count | Number of user-generated transcriptions attached to the representation. | | 1 |
| ^ca_object_representations.page_count | Number of pages in a multipage document. Will be null when not applicable. | | 10 |
| ^ca_object_representations.preview_count | Number of previews available for a multipage document ot timebased media. Will be null when not applicable. This is a synonym for page_count | | 10 |
| ^ca_object_representations.media_dimensions | Pixel dimensions for the media, formatted for display. | version = version to use. Default is original. | 1024p x 2048p |
| ^ca_object_representations.media_duration | Duration of timebased media, formatted for display. | durationFormat = Sets format of duration. Use "delimited" for delimited format, "hms" for hours/minutes/seconds, "hm" for hours/minutes and "seconds" for output in seconds. Default is "hms". version = version to use. Default is original. | 1h 24m 10s |
| ^ca_object_representations.media_class | General type of originally uploaded media. Possible values are "image", "video", "audio", and "document". | | image |
| ^ca_object_representations.media_format | Format of originally uploaded media, for display. | | JPEG, TIFF |
| ^ca_object_representations.media_filesize | File size of media, formatted for display. | version = version to use. Default is original. | 43.2mb |
| ^ca_object_representations.media_colorspace | Colorspace of media, formatted for display. | version = version to use. Default is original. | RGB, CMYK |
| ^ca_object_representations.media_resolution | Pixel resolution of media, formatted for display. May not be available for all file formats | version = version to use. Default is original. | 300ppi |
| ^ca_object_representations.media_bitdepth | Bit depth of media, formatted for display. May not be available for all file formats. | version = version to use. Default is original. | 8bpp |
| ^ca_object_representations.media_center_x | X-coordinate of user-set center crop point for image media. Coordinate is a decimal fraction of the width of the image. | | 0.43 |
| ^ca_object_representations.media_center_y | Y-coordinate of user-set center crop point for image media. Coordinate is a decimal fraction of the width of the image. | | 0.22 |

### 1.26.14 Splitting apart a date range into separate data points

Single date values that are expressed as ranges (e.g. 2000-2018) can be parsed into separate data points for start and end dates. For example, if you wish to export to MS Excel and would like distinct columns for the first and last dates in the range. You can do so with the following syntax:

```
^ca_objects.your_date_element_code%start_as_iso8601=1
^ca_objects.your_date_element_code%end_as_iso8601=1
```

## 1.27 PDF Output

## 1.28 Generating Labels

## 1.29 Tracking current object location

**Note:** The system for location tracking was completely rebuilt for version 1.7.9 with a new, more general, configuration format and additional features. Older configurations should work as before, but the configuration options described below should be used for new setups. To maintain compatibility with future releases consider updating your existing configuration to use the current options.

### 1.29.1 Overview

CollectiveAccess provides a storage location hierarchy to describe the physical locations where collection objects may be located, displayed or stored. Storage locations are records like any other and may be associated with objects using relationships. One can record the current location of an object by simply creating a relationship between object and location.

This arrangement has the advantage of simplicity but comes with serious drawbacks:

- If your objects move often you'll soon have a long list of previous locations, which can make it difficult to figure out what the *current* location is.
- While the current location can be distinguished using a specific relationship type (Eg. "past location" for previous locations and "current location" for the latest location), you must manage setting of these types yourself, which is labor intensive and prone to error.
- Removing previous locations and only recording only a single, current location will make for a simpler and easier to manage display, but means no location history is maintained. For most users, losing location history data is unacceptable.
- Only storage location records may be used to record location. If an object is on loan or exhibition workarounds must be employed, such as dummy "On loan" and "On exhibition" storage locations records.

The history value tracking system (available as of CollectiveAccess version 1.7.9) provides a flexible way to track object locations over time. It can also be used to track other time-varying information such as provenance and current collection. The system employs tracking *policies* to maintain chronologies based upon one or more data elements, and can return full histories as well as current values for any type of record. Tracking of location is the focus in this discussion, but the approaches described here may be applied to other types of time-varying information.

### 1.29.2 Tracking approaches

To handle the range of tracking methodologies required by different types of museum and archival collections CollectiveAccess offers two approaches to location tracking:

- **Workflow-based location tracking.** Current location is recorded for an object across a range of record types representing various related activities, including loans, movements, occurrences (typically representing exhibitions), collections, deaccession and storage location/inventory. The types of records considered part of the tracking workflow, how their dates are established for assembly into a chronology, and how they are displayed within the chronology are specified in a tracking *policy*. Policies are configured in the *app.conf* configuration file using the *history_tracking_policies* described below.
- **Movement-based location tracking.** Location is recorded for objects in related movement records. Each movement record captures details about a specific change in location for one or more objects. The current location for an object is considered to be the location referred to by the most recent movement by date. Movement of entire storage locations within the location hierarchy can be configured to generate a movement record, allowing current location tracking to be based upon both individual object moves and movements of containers and other storage units. Movement-based tracking is more complicated to configure and use, and is only called for when capture of complex metadata (packing, transport, insurance, etc.) about chain of custody and methods used to transition groups of objects between locations is required. This additional documentation comes at the expense of added complexity and data entry, as every movement of an object or group of objects to a new location requires completion of a full movement record.

**Note:** Movement-based tracking is only used when for object location tracking. If tracking non-location values such as provenance, use workflow-based tracking.

### 1.29.3 Configuration

Configuration for workflow and movement-based tracking use the configuration format, with minor differences. Most configuration occurs within the top-level `history_tracking_policies` entry in *app.conf*. Under this entry are two keys, both mandatory:

- **policies** defines all available tracking policies. Most operational configuration resides under this key.

- **defaults** specifies which policy should be used by default for a given *table*. You may define multiple policies per table and declare specific policies be used in various contexts such as user interface bundles. Default policies are a convenience that reduce configuration complexity by setting a standard policy to be overridden as needed.

An example `history_tracking_policies` configuration for workflow-based location tracking is shown below:

```
history_tracking_policies = {
        defaults = {
                ca_objects = current_location
        },
        policies = {
                current_location = {
                        name = _(Current location),
                        table = ca_objects,
                        mode = workflow, # movements or workflow
                        elements = {
                                ca_storage_locations = {
                                        __default__ = {
                                                date = ca_objects_x_storage_locations.
↪effective_date,
                                                setInterstitialElementsOnAdd =␣
↪[effective_date],
                                                useDatePicker = 0,
                                                template =
                                                <l>^ca_storage_locations.hierarchy.
↪preferred_labels.name%delimiter=__</l>  <ifdef code='ca_objects_x_storage_locations.
↪movement_by'> <br>MOVED BY: ^ca_objects_x_storage_locations.movement_by</ifdef>
↪<ifdef code='ca_objects_x_storage_locations.movement_comments'> <br>COMMENTS: ^ca_
↪objects_x_storage_locations.movement_comments</ifdef>,
                                                trackingRelationshipType = related,
                                                restrictToRelationshipTypes =␣
↪[related]
                                        }
                                },
                                ca_occurrences = {
                                        exhibition = {
                                                date = ca_occurrences.exhibition_date,
                                                setInterstitialElementsOnAdd =␣
↪[effective_date],

                                                template =
                                                 <l>^ca_occurrences.preferred_labels.
↪name</l>,
                                        },
                                        __default__ = {
                                                date = ca_objects_x_occurrences.
↪effective_date,
                                                setInterstitialElementsOnAdd =␣
↪[effective_date],

                                                template =
                                                 <l>^ca_occurrences.idno</l> ^ca_
↪occurrences.preferred_labels.name,
                                        }
                                 },
                                 ca_loans = {
                                        __default__  = {
                                                date = ca_loans_x_objects.effective_
↪date,
                                                setInterstitialElementsOnAdd =␣
↪[effective_date],
```

(continues on next page)

(continued from previous page)

```
                                                color = F78B8B,
                                                template = <l>^ca_loans.idno</l> ^ca_
↪loans.preferred_labels (^ca_loans.institution ^ca_loans.date) <ifdef code='ca_loans_
↪x_objects.movement_comments'> <br>COMMENTS: ^ca_loans_x_objects.movement_comments</
↪ifdef>,
                                                restrictToRelationshipTypes = [loan]
                                          }
                                        }
                                }
                        }
                }
}
```

Within the `policies` section are keys for each configured policy. In the example, a single policy with the code `current_location` is defined. Within each policy are entries for `name` (the display name of the policy), `table` (the tables to which this policy applies), `mode` (workflow or movement-based tracking) and `elements`.

`Elements` defines the various types of data tracked by the policy. Each key is a *table* name. Within each table block are entries for types. The special `__default__` type is used to match any type not explicitly listed for the table. In the example the configuration for storage locations (ca_storage_locations) applies to all types of locations. The ca_occurrences entry includes a configuration specifically for occurrences of type "exhibition", and a default configuration for all other types.

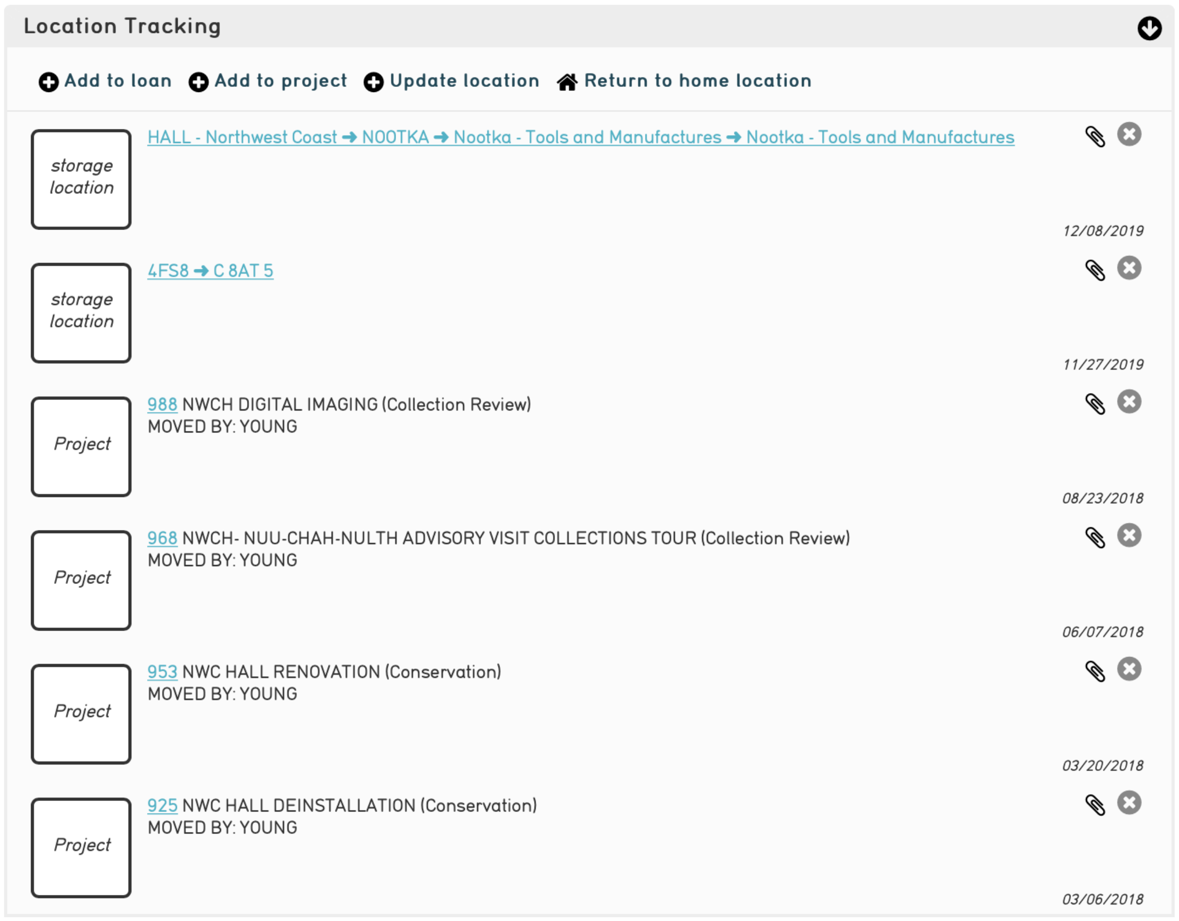
Each per-type configuration must include entries for `date` and `template`. `date` is a bundle specifier for a date field in either the related table or the relationship to that table. The value in the specified field will be used to determine where in the chronology of tracked values each related record is placed. In the example, the object-location relationship `effective_date` intrinsic field is used to track locations, which the occurrence `exhibition_date` metadata element is used to place exhibitions in time. `template` is a *display template* employed to format data for the related record in the chronology. The template will be evaluated relative to the relationship between the object and related record, allowing inclusion of both interstitial (relationship-based) and related-record metadata. In the example the template for loans includes data from both the related loan record as well as the object-loan relationship.

Other, optional keys in per-type configuration configuration include `color` (chronology color-coding), `restrictToRelationshipTypes` (a list of relationship types to limit chronology display to), `setInterstitialElementsOnAdd` (a list of interstitial fields to allow the user to set when creating a relationship from within the chronology). The full list of possible entries is:

| Entry name | Description | Mandatory? |
|---|---|---|
| date | A bundle specified referring to the date metadata element containing the date values used to order related entries in the chronology. | Yes |
| dateMode | When set to "dateless" chronology will display relevant entries in the order in which they were added to CollectiveAccess, most recently added first. If omitted any value other than "dateless", entries will be sorted by their configured date values. | No |
| template | Display template used to format data for the related record in the chronology. The template will be evaluated relative to the relationship between the object and related record, allowing inclusion of both interstitial (relationship-based) and related-record metadata. | Yes |
| setInterstitialElementsOnAdd | A list of interstitial fields to allow the user to set when creating a relationship from within the chronology | No |
| trackingRelationshipType | Relationship type to use by default when creating relationships from within the chronology. | No |
| sortDirection | Direction to order items in the chronology. Default is "DESC" (descending, or most recent first). Use "ASC" to force display with most recent last. | No |
| useDatePicker | Set to non-zero value to enable date picker user interface on date metadata elements. | No |
| restrictToRelationshipTypes | List of relationship types to limit display to in the chronology output. | No |
| color | Color to use for color-coding in the chronology. | No |
| includeFromChildren | If set to a non-zero value, data from child records are included in the chronology. | No |
| childTemplate | If includeFromChildren is set, childTemplate is a display template used to format data from child records. | No |

### 1.29.4 The chronology bundle

You can display a chronology of values for a policy in the editing user interface using the `history_tracking_chronology` bundle.



HALL - Northwest Coast ➜ NOOTKA ➜ Nootka - Tools and Manufactures ➜ Nootka - Tools and Manufactures
12/08/2019

4FS8 ➜ C 8AT 5
11/27/2019

988 NWCH DIGITAL IMAGING (Collection Review)
MOVED BY: YOUNG
08/23/2018

968 NWCH- NUU-CHAH-NULTH ADVISORY VISIT COLLECTIONS TOUR (Collection Review)
MOVED BY: YOUNG
06/07/2018

953 NWC HALL RENOVATION (Conservation)
MOVED BY: YOUNG
03/20/2018

925 NWC HALL DEINSTALLATION (Conservation)
MOVED BY: YOUNG
03/06/2018

The bundle is designed to provide a centralized control panel for managing current location, and includes tools to update location with new loans, movements, occurrences, storage locations, collections and entities. It also offers tools to remove existing relationships and edit interstitial (relationship-specific) data. These tools may be disabled if required.

It displays related locations, occurrences, loans, movements, etc. in chronological order, with the most recent first (although this can be changed). Information from each related record can be formatted using display templates. By default all settings are taken from the policy configuration, but can be overriden by values specific to placements of the bundle in the user interface.

At a minimum when adding a chronology bundle to the editing user interface you must specify a policy. There are many other options which can be set in the an *installation profile* if desired. Available options include:

| Entry name | Description | Mandatory | Default value |
|---|---|---|---|
| policy | The code of history tracking policy to display. | Yes | None |
| displayMode | Format of display. May be set to either "chronology" (chronological view of history) or "tabs" (tabbed display with current value and history separated). Most history management options are only available when using the chronology display mode. | No | chronology |
| dateMode | Determines method used to order the chronology. May be set to either "use_dates" (sort by date as configured) or "dateless" (sort by order entered). "dateless" is typically used as a fallback when legacy location data lacks specific dates. | No | use_dates |
| useAppConfDefaults | Use policy defaults as configured in app.conf when set to a non-zero value. Setting to zero allows override of many options. | No | 1 |
| sortDirection | Direction to order items in the chronology. Default is "DESC" (descending, or most recent first). Use "ASC" to force display with most recent last. | No | DESC |
| currentValueColor | Background color code for current value used when displayMode is "tabs" | No | #EEEEEE |
| futureValueColor | Background color code for future values used when displayMode is "tabs" | No | #EEEEEE |
| pastValueColor | Background color code for historical values used when displayMode is "tabs" | No | #EEEEEE |
| hide_include_child_history_controls | History of children of related records can be included on demand. Set to a non-zero value to hide the tool to control this feature. | No | 0 |
| hide_add_to_loan_controls | Set to a non-zero value to hide the tool for adding loans to the chronology. | No | 0 |
| hide_add_to_movement_controls | Set to a non-zero value to hide the tool for adding movements to the chronology. | No | 0 |
| hide_update_location_controls | Set to a non-zero value to hide the tool for adding storage locations to the chronology. | No | 0 |
| hide_return_to_home_location_controls | Set to a non-zero value to hide the tool for returning an object to its home location | No | 0 |
| hide_add_to_occurrence_controls | Set to a non-zero value to hide the tool for adding occurrences to the chronology. | No | 0 |
| add_to_occurrence_types | A list of occurrence types to offer "add to chronology" options for. Values must be valid occurrence type codes. | No | 0 |
| hide_add_to_collection_controls | Set to a non-zero value to hide the tool for adding collections to the chronology. | No | 0 |
| add_to_collection_types | A list of collection types to offer "add to chronology" options for. Values must be valid collection type codes. | No | 0 |
| hide_add_to_entity_controls | Set to a non-zero value to hide the tool for adding entities to the chronology. | No | 0 |
| add_to_entity_types | A list of entity types to offer "add to chronology" options for. Values must be valid entity type codes. | No | 0 |
| hide_add_to_object_controls | Set to a non-zero value to hide the tool for adding objects to the chronology. | No | 0 |
| useHierarchicalBrowser | Set to a non-zero value to offer a hierarchcy browser when selecting storage locations. If not set a type-ahead lookup will be used. | No | 0 |
| hide_value_interstitial_edit | Set to a non-zero value to hide tools for editing interstitial data on relationships displayed in the chronology. | No | 0 |
| hide_value_delete | Set to a non-zero value to hide the tool from deleting items from the chronology | No | 0 |

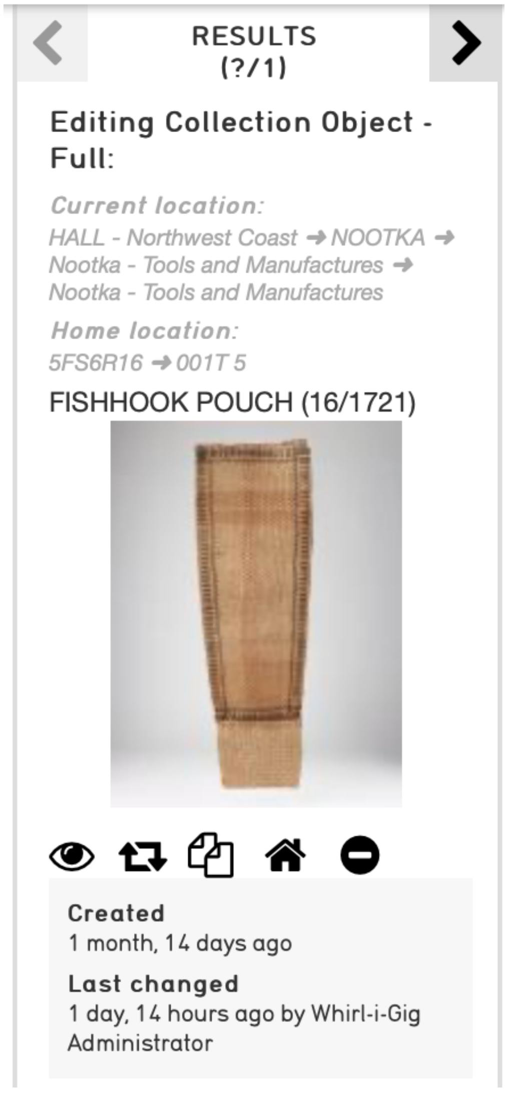
### 1.29.5 The current contents bundle

The current contents bundle (`history_tracking_current_contents`) enables display of all items that currently have a given record as their current value. It is typically used on storage location records to display a list of objects currently resident in that location.

The following options are available to set in an *installation profile*:

| Entry name | Description | Mandatory | Default value |
|---|---|---|---|
| policy | The code of history tracking policy to use for display. | Yes | None |
| list_format | Format of display. May be set to either “bubbles” (small stacked entries) or “list” (a list view of items). | No | bubbles |
| colorItem | Background color to use on items. | No | #FFFFFF |
| displayTemplate | A display template to use for each item displayed | No | The preferred label of the related item |

### 1.29.6 Inspector display



Editing Collection Object - Full:

Current location:
HALL - Northwest Coast ➔ NOOTKA ➔ Nootka - Tools and Manufactures ➔ Nootka - Tools and Manufactures

Home location:
5FS6R16 ➔ 001T 5

FISHHOOK POUCH (16/1721)

Created
1 month, 14 days ago

Last changed
1 day, 14 hours ago by Whirl-i-Gig Administrator

You can display the current value of a history tracking policy in the editor "inspector" (the information panel on the upper left-hand corner of the editor interface). You can set the policy to use on a per-table and/or per-type basis using the `inspector_tracking_displays` entry in app.conf.

```
inspector_tracking_displays = {
        ca_objects = {
                __default__ = {
                        policy = current_location,
                        label = _(Current location)
                }
        }
}
```

Each entry within `inspector_tracking_displays` is a table name. Each table in turn has a list of types (and/or the catch-all `__default__` type that matches type not explicitly configured). Each type has two entries: `policy` (the policy to use) and `label` (A label placed above the current value). In the example above the current value for the "current_location" policy is displayed when editing objects of all types.

A typical inspector with this configuration would appear as show in the screen image on the right.

### 1.29.7 Display in templates

Current value information may be included in *display templates* using the following tags:

| Tag | Description |
|---|---|
| ^<table>.history_tracking_current_value | The current value, formatted using the template as specified in the policy configuration. If multiple policies are defined the default policy is used unless a policy is specified using the policy parameters. (Ex. ^ca_objects.history_tracking_current_value%policy=provenance) |
| ^<table>.history_tracking_current_date | The date of the current value. If multiple policies are defined the default policy is used unless a policy is specified using the policy parameters. (Ex. ^ca_objects.history_tracking_current_date%policy=provenance) |
| ^<table>.history_tracking_current_contents | A list of items whose current value is the subject of the display template. For a current location policy for objects, ^history_tracking_current_contents will display in a template for a storage location (for example) all objects currently at that location. (Ex. ^ca_storage_locations.history_tracking_current_contents)Object preferred labels are returned. The template is not configurable. |

### 1.29.8 Searching on current values

Current values can be indexed for search on a per-table, per-policy basis. Any value in the related table can be indexed, enabling one to search, for example, on the description of current loans only for objects. Typically only basic values such as name and identifier are indexed as current values, allowing for searches on storage location names, loan recipients, etc.

To set up current value indexing you will need to insert new directives into your *search_indexing.conf* file. For each related table block to be indexed add a new `current_values` entry. Within this entry add entries for each policy. Within the policy entry add field indexing entries in the same format as used for regular indexing.

The example below is a fragment from the `ca_objects` indexing configuration. Note the added `current_values` blocks. `current_location` refers to a policy configured in app.conf.

```
# ------------------------------------
ca_storage_locations = {
        tables = {
                places = [ca_objects_x_storage_locations],
        },
        fields = {
                location_id = { DONT_INCLUDE_IN_SEARCH_FORM },
                idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 }
        },
        current_values = {
            current_location = {
                    idno = { STORE, DONT_TOKENIZE, INDEX_AS_IDNO, BOOST = 100 }
                }
        }
},
# ------------------------------------
ca_storage_location_labels = {
        tables = {
                places = [ca_objects_x_storage_locations, ca_storage_locations]
```

(continues on next page)

(continued from previous page)

```
        },
        fields = {
                location_id = { DONT_INCLUDE_IN_SEARCH_FORM },
                name = { INDEX_ANCESTORS, INDEX_ANCESTORS_START_AT_LEVEL = 0, INDEX_
↪ANCESTORS_MAX_NUMBER_OF_LEVELS = 10, INDEX_ANCESTORS_AS_PATH_WITH_DELIMITER = .}
        },
        current_values = {
            current_location = {
                    name = { }
                }
        }
},
# ------------------------------------
```

In this example both the "idno" intrinsic field (part of ca_storage_locations) and the "name" intrinsic field in storage location preferred labels (the ca_storage_location_labels table) are indexed for objects as current values.

To search on current values use the built-in "current_values" access point. Eg. to find all records with current value "Cellar" in any field search on `` `current_values:Celler` ``. To limit the search to a specific policy use the access point "current_values.<policy code>". Eg. `` `current_values.current_location:Cellar` ``. To search on a specific policy and field use "current_values.<policy_code>.<field code>". The field code used must be indexed for the search to return results.

These same access point formats can be used when configuring advanced search forms.

## 1.29.9 Browsing on current values

To browse on current location add a facet to *browse.conf* of type "current_value":

```
current_location = {
                type = current_value,
                restrict_to_types = [],
                policy = current_location,

                display = {
                        ca_storage_locations = {
                                __default__ = { template = ^ca_storage_locations.
↪hierarchy.preferred_labels.name%delimiter=_&gt;_ }
                        }
                },

                include_none_option = No location specified,

                label_singular = _("current location"),
                label_plural = _("current locations")
        },
```

Current value-specific settings include `policy`, which must be set and `display`, which customizes display of current values within the browse. If not defined formatting from the policy is used.

The `collapse` facet option controls which sorts of current values are collapsed into general headings rather than displayed individually. Keys of the entry are table names and type separated with a slash ("/"). Values are text with which to represent the collapsed group in the browse facet. For example, to collapse all occurrences of type "exhibition" into a single facet value labeled "On loan" use:

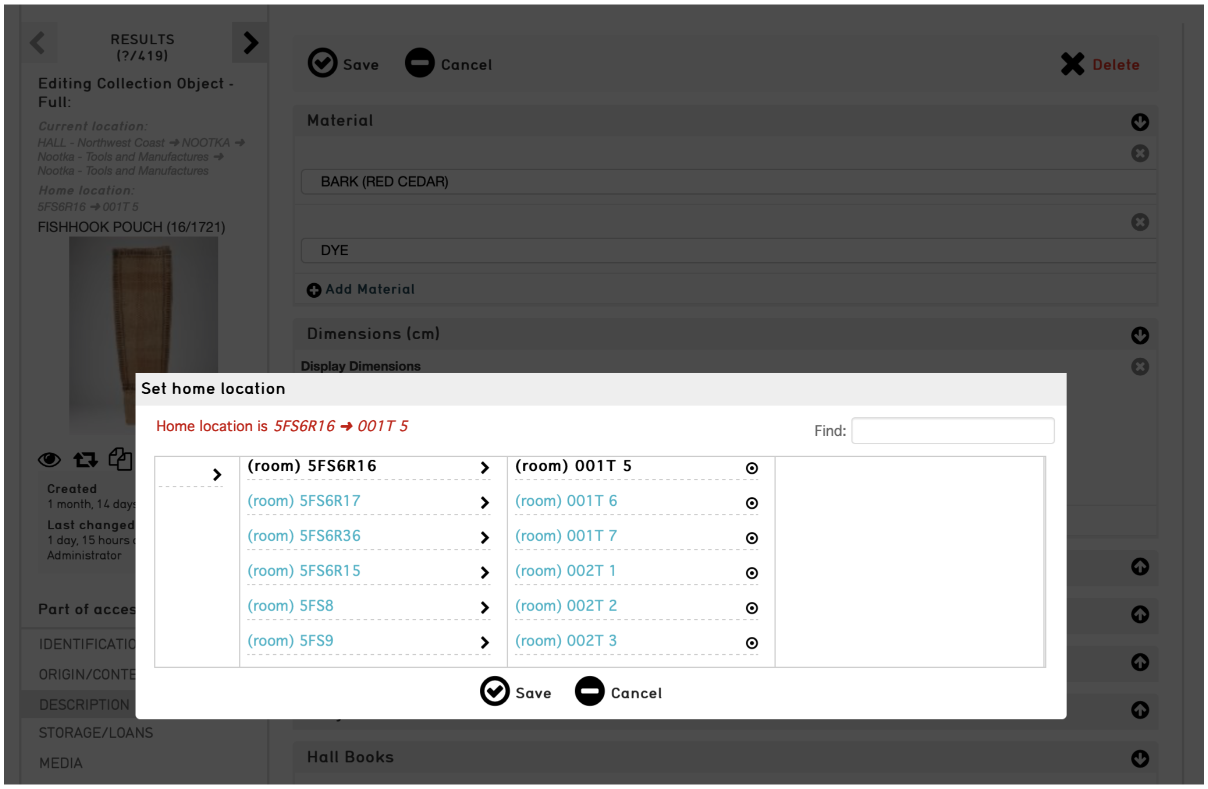
```
collapse = {
        ca_occurrences/exhibition = On loan
}
```

Selecting "On loan" would return all objects where the current location is any exhibition. Without the collapse setting, each exhibition would be listed individually.

## 1.29.10 Home locations

As of version 1.7.9 it is possible to set a "home" location for an object. The home location is its typical storage location. If set, both the chronology (`history_tracking_chronology`) and contents (`history_tracking_current_contents`) bundles can include options to return objects to their home locations, noting the change in the chronology.

Home location can be set by clicking the small house icon in the object editor inspector panel. A hierarchy browser will appear from which you can select the home location.



Set home location

Home location is *5FS6R16 ➔ 001T 5*

Find:

| | | |
|---|---|---|
| | (room) 5FS6R16 | (room) 001T 5 |
| | (room) 5FS6R17 | (room) 001T 6 |
| | (room) 5FS6R36 | (room) 001T 7 |
| | (room) 5FS6R15 | (room) 002T 1 |
| | (room) 5FS8 | (room) 002T 2 |
| | (room) 5FS9 | (room) 002T 3 |

Save Cancel

To display the home location in the inspector panel set the `inspector_home_location_display_template` entry in app.conf to show the desired storage location fields and formatting. The `home_location_display_template` entry defines a template for formatting the home location in display templates and in the hierarchy browser.

A reasonable configuration for these entries, displaying the selected home location prefixed by its parent location is:

```
inspector_home_location_display_template = "<unit relativeTo='ca_storage_locations.
↪hierarchy' delimiter='  '>^ca_storage_locations.preferred_labels.name</unit>"
home_location_display_template = <l><inspector_home_location_display_template></l>
```

The `inspector_home_location_display_template` sets the format in the above example. The `home_location_display_template` takes that format and surrounds it with <l> tags to make it a clickable link.

Home locations can be output in display templates for objects using the tag ``^ca_objects.home_location_value``. The value returned by this tag will be formatted according to the template format in the app.conf `home_location_display_template` entry.

### 1.29.11 Updating the cache

For performance reasons, the current location of the object is cached in the database and used when browsing. Since current location values are calculated based upon the settings in the app.conf change in configuration will likely invalidate the cached data. To regenerate the cache and ensure accurate browse results be sure to run the following caUtils command on the command line:

```
caUtils reload-current-values-for-history-tracking-policies
```

If your current value browse is returning unexpected results it is recommended to run the command, which may resolve the issue.

## 1.30 Workflow-based location tracking

### 1.30.1 Configuration

Unlike the other location tracking options, which only handle storage locations, workflow-based location tracking calculates the current location of an object by looking at a range of related records, including Loans and Occurrences, comparing their dates, and selecting the most recently dated. What types of records are considered and which date elements in those records are used for comparison are entirely configurable.

Workflow-based location tracking can supplement direct object-location reference or movement-based tracking. That is, locations recorded with those two methods may be part of the mix of records workflow-based tracking considers when calculating the current location, but they don't have to be.

#### Step 1

Primary configuration is done in **app.conf** through the **current_location_criteria** directive. current_location_criteria is an associative array the keys of which are the primary types you want considered. Relevant primary types for location tracking are: ca_storage_locations, ca_loans, ca_movements, ca_object_lots and ca_occurrences. Each primary type has a sub-array the keys of which are sub-types (except for ca_storage_locations for which it is a relationship type). Each sub-type/relationship type in turn has an array of options. For example:

```
current_location_criteria = {
   ca_storage_locations = {
      related = {
         template = ^ca_storage_locations.hierarchy.preferred_labels%delimiter=__
      }
   },
   ca_movements = {
      shipping = { date = pickup_date, color = 9bae33 },
      framing = { date = pickup_date, color = 541353 },
      conservation = { date = pickup_date, color = 245442 },
      administrative = { date = pickup_date, color = 992222 },
   },
   ca_loans = {
      collection = {
         date = loan_period,
         color = cccccc
      }
   },
   ca_occurrences = {
      exhibition = {
         date = exh_dates,
         color = 00cc00
      }
   },
   # The entry for ca_objects controls if and how deaccessions are displayed
   ca_objects = {
      template = ^ca_objects.deaccession_notes (^ca_objects.deaccession_date),
      color = cc0000
   }
}
```

In this example, ca_movements is a primary table, while shipping is a movement type and date is an option for the shipping type (and others as well) specifying what date element should be used to calculate this movement types place in the object's history. (For the ca_storage_locations primary type in the example, related is an object-storage location relationship type, and template is an option of that relationship type).

Note that display of deaccessions (managed via the ca_objects_deaccession editor bundle) in the object use history is controlled using the ca_objects primary type. If it is present in the configuration deaccessions will be shown, formatted using the supplied template and color, as in the example above.

Sub-type/relationship type options affect both the what is considered current and how the current location is displayed. Options include:

| Option Name | Description | Mandatory? |
|---|---|---|
| template | A display template evaluated relative to the related record when it is calculated to be current | No. Will default to displaying the preferred label of the record |
| date | The code of the DateRange metadata element used when calculating current location. | Yes for all primary types except ca_storage_locations, for which date is derived from the object-storage location relationship. |
| color | A color to highlight entries of this type with. Should be six-digit hex color | No |

This configuration will be used to display current location in the editor inspectors, when browsing on workflow-based current location and by default in the **Object Use History (ca_objects_history) editor bundle.**

#### Step 2

The **Object Use History (ca_objects_history) editor bundle** bundle is used to display the current location as well as a detailed history of previous use when using Workflow-based tracking (As opposed to **Current Location (ca_objects_location)** which is for Direct object-location tracking). It is intended as a convenient means to show where an object is and has been, but can also be configured to show any set of related records by date. The bundle has a variety of settings to customize the layout and contents of the location stream. All of these can be set in the current_location_criteria bundle in app.conf, described previously, and used as defaults in the bundle. Let's take a look at an example:

In the bundle seen above the cataloguer has configured different colors and templates to showcase Accession, Loan, and Storage Location activity and data. Each block is automatically sorted by the date chosen through the bundle settings for that table. For example, Artwork loans are sorted on the "Loan Period" as seen via the dates on the far right-hand side. When a new relationship is created to any of the three configured tables a new segment will appear in the stream in the appropriate order based on date. In addition to the tables shown in the example, Occurrences, Movements, and Deaccessions can also be configured.

The contents of each block in the stream are entirely configurable using metadata display templates. With this powerful syntax any metadata from the related record, or from those records related to the related record, can be displayed in the Use History bundle. An example of that relationship traversing can be seen above in the Artwork loan blocks. There, the "Borrower" is displayed using the below syntax which pulls entities related to the related loan:

```
<l>^ca_loans.preferred_labels</l><br>
<ifdef code="ca_loans.loan_period">Loan Period:</ifdef> ^ca_loans.loan_period<br>
Borrower: <unit relativeTo="ca_loans">
<unit relativeTo="ca_entities" delimiter=", " restrictToRelationshipTypes="borrower">^
↪ca_entities.preferred_labels</unit></unit>
```

### 1.30.2 Configuring bundle-specific settings through an installation profile

To add the Use History bundle to the installation profile, simply include the bundle placement and relevant settings on the appropriate UI screen. The use history settings defined in app.conf are taken as a system-wide universal, but defining the ca_objects_history setting in the profile allows for UI-specific customizations.

```
<placement code="ca_objects_history">
  <bundle>ca_objects_history</bundle>
  <settings>
    <setting name="ca_object_lots_purchase_dateElement">accession_date</setting>
    <setting name="ca_object_lots_purchase_color">#663A8C</setting>
  </settings>
</placement>
```

The chart below lists settings per table that can be included in your profile. Be sure to replace #type# with the custom type configured in your profile. For example, if "purchase" was the item idno in your list ca_object_lot_types, then your setting would be: ca_object_lots_purchase_dateElement.

Note that there is no dateElement setting for storage locations. Storage locations are sorted on the date cataloged.

| Setting Name | Description |
|---|---|
| hide_add_to_loan_controls | Set to 1 if you want to to hide the "Add to loan" controls in this bundle placement. Defaults to 0 |
| hide_update_location_controls | Set to 1 if you want to to hide the "Update Location" controls in this bundle placement. Defaults to 0 |
| useHierarchicalBrowser | Set to 1 if you want to provide an hierarchical browser when searching for an updated storage location |
| locationTrackingMode | Sets method for tracking location of object within storage location hierarchy. Set to ca_storage_locations to use direct object-location references; set to ca_movements to use movement-based location tracking |
| useAppConfDefaults | Set to 1 if you want to use the app.conf current_location_criteria settings. If you want to set specific settings for this bundle then set this to 0 and specify values for that various settings below |
| ca_object_lots_showTypes | Sets which object lots types appear in the use history stream. For all showTypes settings, repeat once for each type that should be included. Use the item idno i.e. <setting name="ca_object_lots_showTypes">purchase</setting> |
| ca_object_lots_#type#_dateElement | Sets the date field that is used to sort object lots of this type in the use history stream. |
| ca_object_lots_#type#_color | Sets the color for this type of object lot. Use hex colors with no pound sign, i.e. "B1AAF2" |
| ca_object_lots_#type#_displayTemplate | Sets the template for metadata displayed for this type of object lot in the use history stream. |
| ca_occurrences_showTypes | Sets which occurrence types appear in the use history stream. |
| ca_occurrences_#type#_dateElement | Sets the date field that is used to sort this type of occurrence in the use history stream. |
| ca_occurrences_#type#_color | Sets the color for this type of occurrence. Use hex colors with no pound sign, i.e. "B1AAF2" |
| ca_occurrences_#type#_displayTemplate | Sets the template for metadata displayed for this type of occurrence in the use history stream |
| ca_movements_showTypes | Sets which movement types appear in the use history stream. |
| ca_movements_#type#_dateElement | Sets the date field that is used to sort this type of movement in the use history stream. |
| ca_movements_#type#_color | Sets the color for this type of movement. Use hex colors with no pound sign, i.e. "B1AAF2" |
| ca_movements_#type#_displayTemplate | Sets the template for metadata displayed for this type of movement in the use history stream |
| ca_loans_showTypes | Sets which loan types appear in the use history stream. |
| ca_loans_#type#_dateElement | Sets the date field that is used to sort this type of loan in the use history stream. |
| ca_loans_#type#_color | Sets the color for this type of loan. Use hex colors with no pound sign, i.e. "B1AAF2" |
| ca_loans_#type#_displayTemplate | Sets the template for metadata displayed for this type of loan in the use history stream |
| ca_storage_locations_showRelationshipTypes | Sets which related storage locations will appear. |
| ca_storage_locations_color | Sets the color for all storage locations types. Use hex colors with no pound sign, i.e. "B1AAF2" |
| ca_storage_locations_displayTemplate | Sets the template for metadata displayed for storage locations of all types. |
| showDeaccessionInformation | Sets whether or not Deaccessions are included in the stream. 1 = yes; 0 = no. |
| deaccession_color | Sets the color for deaccessions. Use hex colors with no pound sign, i.e. "B1AAF2" |
| deaccession_displayTemplate | Sets the template for deaccessions. |

### 1.30.3 Browsing by current location

Workflow-based location tracking will cache the current location of the object within the object record, which makes browsing possible. To set up a current location browse add a facet of type location in browse.conf. For example:

```
current_location = {
  type = location,
   restrict_to_types = [],

   group_mode = none,

   collapse = {
      ca_loans = On loan,
      ca_movements/conservation = In conservation,
      ca_movements/shipping = Shipped,
      ca_movements/administrative = Consigned
   },

   display = {
      ca_storage_locations = {
         related = { template = ^ca_storage_locations.hierarchy.preferred_labels
↪%delimiter=__ (storage) }
      },
      ca_occurrences = {
         exhibition = { template = ^ca_occurrences.preferred_labels.name (exhibition)␣
↪}
      },
   },
   maximumBrowseDepth = 1,
   include_none_option = No location specified,

   label_singular = _("current location"),
   label_plural = _("current location")
}
```

The collapse, display, maximumBrowseDepth and include_none_option directives are specific to location facets:

| Directive | Description | Mandatory? |
|---|---|---|
| collapse | The facet will list entries for each distinct storage location. By default this means the each individual loan, movement, storage location or occurrence will be listed in the facet. For storage locations and occurrences this usually exactly what you want. Listing individual loans and movements, however, often results in a long list of undifferentiated values that makes effective browsing difficult. collapse' directs the browse to consolidate all records of a given type (and optionally) sub-type into a single generic facet value. In the example above, collapsing onca_loansresults in all objects for which the current location is a loan being browsable via a single "On loan" facet item. For movements, facet items are presented for individual movement sub-types. | No. |
| display | Sets the display template used to generate the facet item for the specified type and sub-type (or in the case of storage locations, type and relationship type). Each template is generated relative to the related record. | No. |
| include_none_option | Enable browsing for objects that have no current location set. The values of this directive will be used as the text of the facet item. | No |
| maximumBrowseDepth | As of version 1.7: Enables the bucketing of storage locations into top-level facets. Child locations are grouped by their parent, based on the depth set through this configuration. A browse at the parent level will be inclusive of all child locations. For example, objects currently stored in the locations: Offsite Storage > Room 1 and Offsite Storage > Room 2 will return when a user browses on Offsite Storage if the maximumBrowseDepth is set to 1. | No |

### 1.30.4 Updating the cache

For performance reasons, the current location of the object is cached within the object record itself. Since locations are calculated based upon the settings in the app.conf current_location_criteria directive, and change in current_location_criteria will likely invalidate the cached data. To regenerate the cache and ensure accurate browse results be sure to run the following caUtils command on the command line:

```
bin/caUtils reload-object-current-locations
```

### 1.30.5 General maintenance

Both direct object-location and movement-based location tracking rely on dates embedded in relationships between related records. If you are updating an older system, change app.conf configuration or otherwise have reason to believe these dates may be out of sync with the underlying movement and location data from which they are derived you can run the following caUtils command on the command line to refresh values:

```
bin/caUtils reload-object-current-location-dates
```

For most data sets this command should take only seconds to a few minutes to run and will not have adverse effects. If you are getting odd ordering in use histories or display of current location try running this command to resolve the issues.

### 1.30.6 Displaying current locations in reports

As of version 1.6 an object's current location can be included in reports via the Displays editor. To include the location, simply drag the "Current Location" bundle (also shown as "Object Location") onto your Display.

By default this bundle will display the Current Location as it is defined by the current_location_criteria (see above). Put another way, the report will output the same formatting used for location tracking in the cataloging interface. To override this formatting, use the "display format" setting on the "Object Location" bundle. To include the activity date use the syntax: ^ca_objects.ca_objects_location_date. To show the current_location_criteria use the syntax: ^ca_objects.ca_objects_location.

## 1.31 Import Mappings

### 1.31.1 Rules

### Rule description

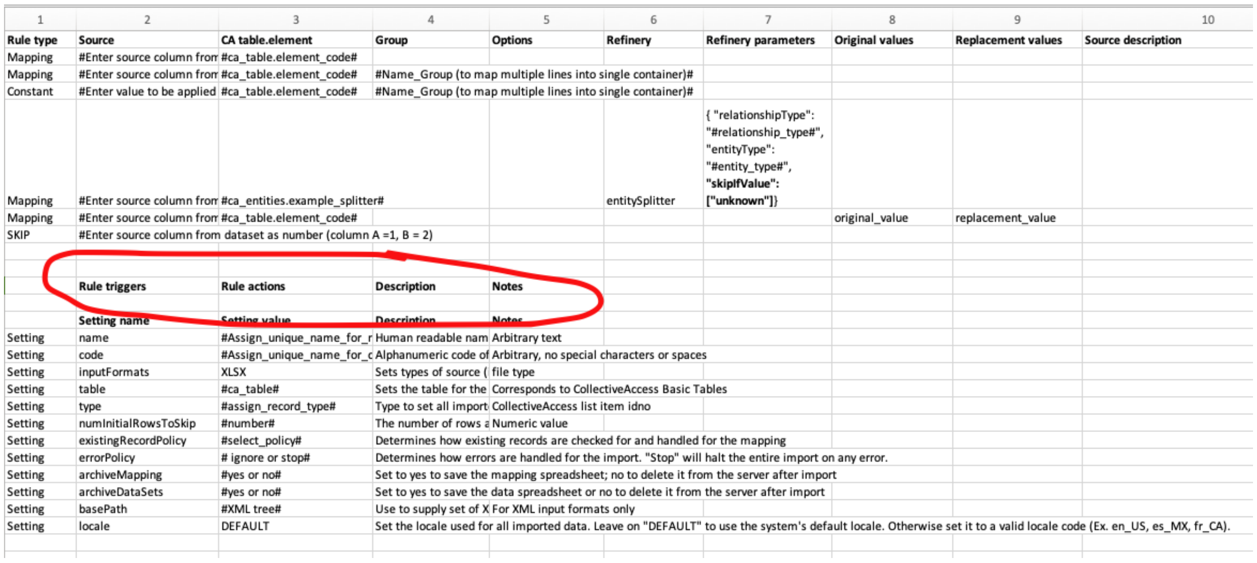
Rules allow you to set record-level conditionals in your mapping with target actions triggered by true or false outcomes. With Rules, you can manipulate the migration of specific data and/or set metadata based on expression statements. For example, let's say you want to skip a record if a certain element in your data source is exactly equal to a specific value. Rules allows you to set a target action, such as "SKIP," when a match is triggered.

Rules rely on a two part operation outlined in the import mapping. The first component is is called "Rule triggers" and it is an expression statement that results in a quantity that is evaluated by the data importer. The second defines "rule actions" that are performed based on the outcome of the expression.

Let's walk through an example. (See also the in-depth description of *expressions*).

For our example, we are going to skip all records with the phrase "do not use" in the description. To do so we write an expression to match "do not use" in the required field, and then set the action to execute when the expression is true to be "SKIP." For the sake of this example we're importing an Excel spreadsheet and the description is in column 5:

This is how the rule should look in the import mapping:



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| **Rule type** | **Source** | **CA table.element** | **Group** | **Options** | **Refinery** | **Refinery parameters** | **Original values** | **Replacement values** | **Source description** |
| Mapping | #Enter source column fron | #ca_table.element_code# | | | | | | | |
| Mapping | #Enter source column fron | #ca_table.element_code# | #Name_Group (to map multiple lines into single container)# | | | | | | |
| Constant | #Enter value to be applied | #ca_table.element_code# | #Name_Group (to map multiple lines into single container)# | | | | | | |
| Mapping | #Enter source column fron | #ca_entities.example_splitter# | | | entitySplitter | { "relationshipType": "#relationship_type#", "entityType": "#entity_type#", **"skipIfValue": ["unknown"]**} | | | |
| Mapping | #Enter source column fron | #ca_table.element_code# | | | | | original_value | replacement_value | |
| SKIP | #Enter source column from dataset as number (column A =1, B = 2) | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | **Rule triggers** | **Rule actions** | **Description** | **Notes** | | | | | |
| | | | | | | | | | |
| | **Setting name** | **Setting value** | **Description** | **Notes** | | | | | |
| Setting | name | #Assign_unique_name_for_r | Human readable nam | Arbitrary text | | | | | |
| Setting | code | #Assign_unique_name_for_c | Alphanumeric code of | Arbitrary, no special characters or spaces | | | | | |
| Setting | inputFormats | XLSX | Sets types of source ( | file type | | | | | |
| Setting | table | #ca_table# | Sets the table for the | Corresponds to CollectiveAccess Basic Tables | | | | | |
| Setting | type | #assign_record_type# | Type to set all import | CollectiveAccess list item idno | | | | | |
| Setting | numInitialRowsToSkip | #number# | The number of rows a | Numeric value | | | | | |
| Setting | existingRecordPolicy | #select_policy# | Determines how existing records are checked for and handled for the mapping | | | | | | |
| Setting | errorPolicy | # ignore or stop# | Determines how errors are handled for the import. "Stop" will halt the entire import on any error. | | | | | | |
| Setting | archiveMapping | #yes or no# | Set to yes to save the mapping spreadsheet; no to delete it from the server after import | | | | | | |
| Setting | archiveDataSets | #yes or no# | Set to yes to save the data spreadsheet or no to delete it from the server after import | | | | | | |
| Setting | basePath | #XML tree# | Use to supply set of X | For XML input formats only | | | | | |
| Setting | locale | DEFAULT | Set the locale used for all imported data. Leave on "DEFAULT" to use the system's default locale. Otherwise set it to a valid locale code (Ex. en_US, es_MX, fr_CA). | | | | | | |

Set "Rule" as your rule type and add the following to the Rule triggers column:

(^5 =~ /do not use/)

Where ^5 references column 5 and =~ invokes the regular expression operator. In the "actions" column is a simple reference to the action:

SKIP

Note that you can potentially add several actions to a single rule trigger by separating the actions with returns. For CollectiveAccess versions up to 1.6, the only possible action is "SKIP", which skips the entire record rather than importing it.

From version 1.7 a "SET" action is also available, allowing the injection of arbitrary values into the import source data subject to trigger criteria. These values may then be mapped as if they were actually present in the original source data.

SET actions are more complicated than SKIP, requiring more than just the action name to be defined. To accommodate this, the actions list must be in JSON format for SET. The block is a list of objects, each of which has four keys:

| Key | Description |
|---|---|
| action | Name of action - SET in this case |
| target | The name of the injected value. You can use this name is mappings as you would use any other element in the source data. Eg. if you set this to inventory_yn_value then you will be able to map inventory_yn_value as a source to any CollectiveAccess element. |
| value | The value to inject into the source data under the target name when the trigger evaluates as true. |
| else | The value to inject into the source data under the target name when the trigger evaluates as false. |

An action list with a single SET action looks like this:

```
[{
 "action": "SET",
 "target": "inventory_yn_value",
 "value": "yes",
 "else": "no"
}]
```

### 1.31.2 Refineries

https://docs.collectiveaccess.org/wiki/Data_Importer#Refineries

### 1.31.3 Builders

- *CollectionHierarchyBuilder*
- *CollectionIndentedHierarchyBuilder*
- *EntityHierarchyBuilder*
- *PlaceHierarchyBuilder*
- *ObjectHierarchyBuilder*
- *OccurrenceHierarchyBuilder*
- *ListItemHierarchyBuilder*
- *ListItemIndentedHierarchyBuilder*
- *StorageLocationHierarchyBuilder*

#### CollectionHierarchyBuilder

DOCS at https://docs.collectiveaccess.org/wiki/Data_Importer#Refineries

#### CollectionIndentedHierarchyBuilder

NEW, built for Apollo collections import in 2019 for Alex, Seth to document

#### EntityHierarchyBuilder

#### PlaceHierarchyBuilder

#### ObjectHierarchyBuilder

#### OccurrenceHierarchyBuilder

#### ListItemHierarchyBuilder

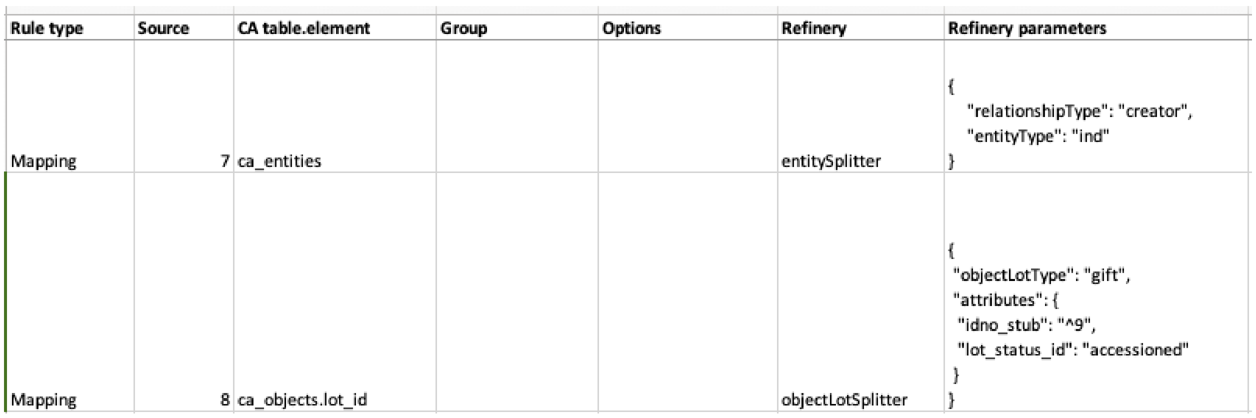
#### ListItemIndentedHierarchyBuilder

#### StorageLocationHierarchyBuilder

## 1.31.4 Splitters

### Import Mapping

See below for examples of splitters in an import mapping:



| Rule type | Source | CA table.element | Group | Options | Refinery | Refinery parameters |
|---|---|---|---|---|---|---|
| Mapping | 7 | ca_entities | | | entitySplitter | { "relationshipType": "creator", "entityType": "ind" } |
| Mapping | 8 | ca_objects.lot_id | | | objectLotSplitter | { "objectLotType": "gift", "attributes": { "idno_stub": "^9", "lot_status_id": "accessioned" } } |

## CA table.element

Enter the splitter in the Refinery column of the import mapping. Enter the CollectiveAccess table as shown in the chart below.

| CA table.element | Refinery |
|---|---|
| ca_collections | collectionSplitter |
| ca_entities | entitySplitter |
| ca_list_items | listItemSplitter |
| ca_loans | loanSplitter |
| NO TABLE | measurementsSplitter |
| ca_movements | movementSplitter |
| ca_places | placeSplitter |
| ca_objects | objectSplitter |
| ca_objects.lot_id | objectLotsSplitter |
| ca_ocurrences | occurrenceSplitter |
| ca_tour_stops | tourStopSplitter |
| ca_storage_locations | storageLocationSplitter |

## Refinery Options

Splitter refineries can either create records or match data to existing records (following a mapping's existingRecordPolicy) or break a single string of source data into several metadata elements in CollectiveAccess. Splitters for relationships are used when several parameters are required, such as setting a record type and setting a relationship type. Using the entitySplitter, a name in a single location (i.e. column) in a data source can be parsed (into first, middle, last, prefix, suffix, et al.) within the new record. Similarly the measurementSplitter breaks up, for example, a list of dimensions into to a CollectiveAccess container of sub-elements. "Splitter" also implies that multiple data elements, delimited in a single location, can be "split" into unique records related to the imported record. Refinery options are listed below.

## attributes

Sets or maps metadata for the entity record by referencing the metadataElement code and the location in the data source where the data values can be found

See below for additonal attribute settings for the entitySplitter and objectRepresentationSplitter

**Example**

```
{"attributes": {
   "address": {
      "address1": "^24",
      "address2": "^25",
      "city": "^26",
      "stateprovince": "^27",
      "postalcode": "^28",
      "country": "^29"
   }
 }
        }
```

### Additional Properties - Auto-generated idnos

To map source data to idnos, see the 'attributes' parameter above. An exception exists for when idnos are set to be auto-generated. To create auto-generated idnos within any splitters where it's relevant (i.e measurementSplitter doesn't support this), use the following syntax.

`"attributes":  {"idno":"%"}`

### objectRepresentationSplitter Additional Properties

Sets the attributes for the object representation. "Media" sets the source of the media filename in the data, which is what will match on the actual media file in the import directory. Note: filenames in source data may or may not the include file extension, but source data must match filename exactly. Set the media filename to idno, using "idno". Additional attributes, such as the example, "internal_notes", can also be set here.

```
{"attributes":{
   "media": "^1",
   "internal_notes": "^2",
   "idno": "^1"
}
}
```

*Applicable refineries*: collectionSplitter, entitySplitter, listItemSplitter, loanSplitter, measurementsSplitter, movementSplitter, placeSplitter, objectSplitter, objectLotsSplitter, occurrenceSplitter, tourStopSplitter

### Type

Accepts a constant list item idno from the list (collection_types, object_types, entity_types, list_item_types, loan_types) or a reference to the location in the data source where the type can be found

| Splitter | Type |
|---|---|
| collectionSplitter | collectionType |
| entitySplitter | entityType |
| listItemSplitter | listItemType |
| loanSplitter | loanType |
| measurementsSplitter | |
| movementSplitter | |
| placeSplitter | |
| objectSplitter | |
| objectLotsSplitter | |
| occurrenceSplitter | |
| tourStopSplitter | |
| storageLocationSplitter | |

*Applicable Refineries*: collectionSplitter, entitySplitter, listItemSplitter, loanSplitter

### TypeDefault

Sets the default type that will be used if none are defined or if the data source values do not match any values in the CollectiveAccess list types (collection_types, object_types, entity_types, list_item_types, loan_types).

| Splitter | TypeDefault |
|---|---|
| collectionSplitter | collectionTypeDefault |
| entitySplitter | entityTypeDefault |
| listItemSplitter | listItemTypeDefault |
| loanSplitter | loanTypeDefault |
| measurementsSplitter | |
| movementSplitter | |
| placeSplitter | |
| objectSplitter | |
| objectLotsSplitter | |
| occurrenceSplitter | |
| tourStopSplitter | |
| storageLocationSplitter | |

*Applicable Refineries*: collectionSplitter, entitySplitter, loanSplitter, listItemSplitter

### delimiter

Sets the value of the delimiter to break on, separating data source values

```
{"delimiter":  ";"}
```

*Applicable Refineries*: collection Splitter, entitySplitter, listItemSplitter, loanSplitter, measurementsSplitter, movementSplitter, placeSplitter, objectSplitter, objectLotSplitter, objectRepresentationSplitter, occurrenceSplitter, tourStopSplitter

### displayNameFormat

Allows you to format the output of the displayName. Options are: “surnameCommaForename” (forces display name to be surname, forename); “forenameCommaSurname” (forces display name to be forename, surname); “forenameSurname” (forces display name to be forename surname); “original” (is the same as leaving it blank; you just get display name set to the imported text). This option also supports an arbitrary format by using the sub-element codes in a template, i.e. “^surname, ^forename ^middlename”. Doesn’t support full format templating with <unit> and <ifdef> tags, though.

`{"displaynameFormat": "surnameCommaForename"}`

*Applicable Refineries*: entitySplitter

### dontCreate

If set to true (or any non-zero value) the splitter will only do matching and will not create new records when matches are not found.

`{"dontCreate": "1"}`

*Applicable Refineries*: collectionSplitter, entitySplitter, listItemSplitter, loanSplitter, movementSplitter, objectLotsSplitter, objectRepresentationSplitter, objectSplitter, occurrenceSplitter, placeSplitter, tourStopSplitter

### elements

Maps the components of the dimensions to specific metadata elements

```
{"elements": [
   {
      "quantityElement": "measurementWidth",
      "typeElement": "measurementsType",
      "type": "width"
   },
   {
      "quantityElement": "measurementHeight",
      "typeElement": "measurementsType2",
      "type": "height"
   }
]}
```

Note: the typeElement and type sub-components are optional and should only be used in measurement containers that include a type drop-down.

*Applicable Refineries*: measurementsSplitter

### ignoreParent

For use with collection hierarchies. When set to true this parameter allows global match across the entire hierarchy, regardless of parent_id. Use this parameter with datasets that include values to be merged into existing hierarchies but that do not include parent information. Paired with matchOn it’s possible to merge the values using only name or idno, without any need for hierarchy info. Not ideal for situations where multiple matches can not be disambiguated with the information available.

`{"ignoreParent": "1"}`

*Applicable Refineries*: collectionSplitter, entitySplitter, listItemSplitter, loanSplitter, movementSplitter, objectLotsSplitter, objectSplitter, occurrenceSplitter, placeSplitter, tourStopSplitter

### interstitial

Sets or maps metadata for the interstitial movementRelationship record by referencing the metadataElement code and the location in the data source where the data values can be found.

```
{
   "interstitial": {
      "relationshipDate": "^4"
   }
}
```

*Applicable Refineries*: collectionSplitter, entitySplitter, listItemSplitter, loanSplitter, movementSplitter, objectLotsSplitter, objectSplitter, occurrenceSplitter, placeSplitter, tourStopSplitter

### list

Enter the list_code for the list that the item should be added to. This is mandatory - if you forget to set it or set it to a list_code that doesn't exist the mapping will fail.)

`{"list":  "list_code"}`

*Applicable Refineries*: listItemSplitter

### matchOn

From version 1.5. Defines exactly how the splitter will establish matches with pre-existing records. You can set the splitter to match on idno, or labels. You can also include both labels and idno in the matchOn parameter, and it will try multiple matches in the order specified.

"`{""matchOn"":  [""labels"", ""idno""]}` -Will try to match on labels first, then idno.

`{""matchOn"":  [""idno"", ""labels""]}` - Will do the opposite, first idno and then labels.

You can also limit matching by doing one or the other. Eg:

{""matchOn"": ""idno""]} will only match on idno.

{""matchOn"": [""^ca_collections.your_custom_code""]} will match on a custom metadata element in the collection record. Use the syntax ^ca_collections.metadataElement code."

## 1.31.5 Mapping Options

Options allow you to set additional formatting and conditionals on data during import. Some Options are designed to actually set parameters on the mapping behavior, such as the skip options. skipGroupIfEmpty, for example, allows you to prevent the import of certain fields, depending on the presence of data in another related field. Other Options simply format data, such as formatWithTemplate, suffix, and convertNewlinesToHTML.

#### Option: applyRegularExpressions

This option allows the user to effectively rewrite messy and problematic source data using Perl compatible regular expressions as supported in the PHP programming language. Let's say you are mapping duration data to a TimeCode element, and the source data syntax is invalid. See the *Regular Expressions* page for useful regular expressions.

> Invalid timecode format: `7.30.`
>
> Should be transformed to valid timecode format: `7:30`

The invalid data can be transformed using the applyRegularExpressions option with the proper regular expressions.

```
{
   "applyRegularExpressions": [
       {
           "match": "([0-9]+)\\.([0-9]+)",
            "replaceWith": "\\1:\\2"
       },
        {
           "match": "[^0-9:]+",
           "replaceWith": ""
       }
    ]
}
```

match: A regular expression applied to source data values. replaceWith: If a match is found, it will be replaced with whatever is contained in "replaceWith".

In this example, the first regular expression matches <number>.<number> and replaces it with <number>:<number>. In other words, "7.30." becomes "7:30.". The [0-9]+ string matches sequences of 1 or more numbers. Since they're in parenthesis they can be "back referenced" into the replaceWith part using the \1 and \2 placeholders. The second regular expression matches any character that is not a number or a colon (the first one having reformatted any period between numbers as a colon) and replaces it with nothing – removing it in other words. This regular expression takes care of the erroneous period at the end of the invalid data."7:30." is transformed into "7:30" - a valid TimeCode input.

> **Note:** The only deviation from the standard regular expressions language are the backslashes. Wherever you would use a single backslash in a regular expression, you need to use two in our mapping because JSON treats backslashes specially and demands that a literal \ be encoded as \\

#### Option: prefix

#### Option: suffix

#### Transform Values Using Worksheet

Using *Original values and Replacement values (Columns 8-9) <import/mappings:Original values and Replacement values (Columns 8-9)>* is sufficient for transforming a small range of values. But for large transformation dictionaries, use the option "transformValuesUsingWorksheet" instead. You can use this option to reference a list of values in a separate worksheet within the mapping document. The formatting of the sheet should place original values in the first column, and replacement values in the second column.

When this option is set, any values in the "original values" and "replacement values" columns of the mapping worksheet are ignored, even if the "transformValuesUsingWorksheet" worksheet is empty or does not exist. You refer to the sheet by name:

```
{"transformValuesUsingWorksheet":"Worksheet Title"}
```

### 1.31.6 Supported File Formats

#### Overview

Data may be imported into CollectiveAccess in a range of formats, including from Excel, CSV, a range of XML formats and others including external databases such as WorldCat. The fields from these sources are matched to CollectiveAccess tables and fields using the Mapping document's "Source" column, described here.

This page provides an overview of formats compatible with data import as well as how to identify a specific element from the source file for the Mapping document.

#### Currently Supported File Formats

##### XLS, XLSX, CSV, TSV

Spreadsheets are mapped by column, with numeric identifiers provided in the Source column of the import mapping. If you wish to map from Column B of an Excel spreadsheet, you would list the Source as 2. (A = 1, B = 2, C = 3, and so on.)

##### XML (Including FileMaker XML, Inmagic XML, EAD XML, PastPerfect XML, Vernon XML, TEI XML, PBCore XML, MediaBin, MARCXML & MODS)

XML sources are referenced using xPath, a query language for selecting nodes and computing values from XML documents (a basic tutorial is available from W3C).

In general the Source column should be set to the name of the XML tag, proceeded with a forward slash (i.e. /Sponsoring_Department or /inm:ContactName)

Common examples of xPath expressions are provided in the table below.

| Imports | Description | XML example | Mapping source syntax | Notes |
|---|---|---|---|---|
| Items | Imports element items | `<text>`<br>`    <body>`<br>`        <div>`<br>`            ITEM`<br>`        </div>`<br>`    </body>`<br>`</text>` | `/text/body/div` | Example imports "ITEM". |
| Items with a particular attribute value | Imports items only from elements with a certain attribute node value. | `<text>`<br>`    <body>`<br>`        <div attribute="thistype!">`<br>`            ITEM`<br>`        </div>`<br>`    </body>`<br>`</text>` | `/text/body/div[@attribute='thistype!']` | Maps element items with "thistype!" as attribute value. |
| Attribute value | Imports the attribute node value itself. | `<text>`<br>`    <body`<br>`        <div attribute="thistype!">`<br>`            ITEM`<br>`        </div>`<br>`    </body>`<br>`</text>` | `/text/body/div/@attribute` | Using the example, this mapping would import 'thistype!' itself, as opposed to "ITEM". |
| Items *not* of a particular attribute value | Imports items in cases where the element does not have an attribute, or in cases where the attribute value is empty. | `<text>`<br>`    <body>`<br>`       <div>`<br>`            ITEM_1`<br>`        </div>`<br>`        <div attribute="thistype!">`<br>`            ITEM_2`<br>`        </div>`<br>`    </body>`<br>`</text>` | `/text/body/div[not(@attribute)]` | This only necessary in cases where there are other instances where the same element does have the attribute, like the example here. In this case, ITEM_1 would be imported, and ITEM_2 would not. |

MARCXML files can also be imported using the xPath syntax. Standard fields and indicators can be selected as Sources as follows:

| Imports | Description | XML example | Mapping source syntax (XPATH) |
|---|---|---|---|
| 245 - Title Statement | Mapping source for MARC data field 245 subfield a. | `<datafield ind1="1" ind2="4" tag="245">` `<subfield code="a">The human factor /</subfield>` `</datafield>` | `/datafield[@tag='245']/subfield[(@code='a')]` |
| 001- Control Number | Mapping source for MARC control field 001 | `<controlfield tag="001">3780733</controlfield>` | `/controlfield[@tag='001']` |
| 100 - Main Entry: Personal Name | Mapping source for MARC data field 100 subfield a. | `<datafield ind1="1" ind2=" " tag="100">` `<subfield code="a">Greene, Graham,</subfield>` `</datafield>` | `/datafield[@tag='100']/subfield[(@code='a')]` |

FileMakerPro XMLRESULT files generally follow the XML and xPath conventions described above but require some special formatting considerations due to inclusion of invalid characters in field names in certain databases (i.e. ArtBase). Source field names in the mapping must follow these rules:

- Field name should be preceeded with a forward slash (i.e. /Inventory::ArtistLast)
- The importer does not trim trailing spaces in field names so watch out for that!
- Only A-Z a-z 0-9 and these special characters are accepted _ - & # ? % :
- For all other special characters, including a space, replace the character with a single _ (underscore).
- If two invalid special characters fall in a row, use only a single _ (underscore) rather than two

### MARC

MARC records, in addition to MARCXML files, can be imported by using. . .

### EXIF, IPTC, XMP

These embedded metadata standards can be imported from uploaded media (images, video, audio, etc.) using the same import mappings as described above. The *inputFormat* should always be set to "EXIF".

**Note: SYSTEM REQUIREMENT**

To import EXIF data your server must have the free ExifTool application installed on your server. Make sure the ExifTool entry in your external_application.conf configuration file is set to point to the installed application.

EXIF data can be difficult to decifer and locate the desired fields for import as the labels that appear in applications such as Photoshop that use the data often do not match the names given in the underlying EXIF file.

These names can be found by running the ExifTool command-line application. Once installed it can be run as:

```
exiftool -json -a -gl my_file.tiff
```

This will return a set of JSON encoded metadata, which matches the format used by the CollectiveAccess importer, allowing the names of fields within the metadata to be accurately identified. For example this block of EXIF metadata can be used to identify the type of lens used for a photograph:

```
"XMP-aux": {
   "SerialNumber": 1260413208,
   "LensInfo": "18-55mm f/?",
   "Lens": "18.0-55.0 mm",
   "ImageNumber": 0,
   "ApproximateFocusDistance": 4294967295,
   "FlashCompensation": 0,
   "OwnerName": "Erik Garcia Gomez",
   "Firmware": "1.1.1"
},
```

To extract the lens information the block heading "XMP-aux" would be joined with the sub-section "Lens" with a slash to create "XMP-aux/Lens". This would be added to the Source column of the import mapping and matched with a target field in CollectiveAccess.

As this import format is used frequently in conjunction with media import, two more options are available to help identify uploaded media and match metadata to the correct files within the system. Use *_filename_* as a source if you wish to set any field in CollectiveAccess as the filename. And more importantly, *_filepath_* points to the media in the import directory, and can be used to trigger ingestion of the media itself.

| Source | Description | Parameter notes |
|---|---|---|
| __filename__ | This source value takes the filename of the media being imported. You can import filenames to any field in CollectiveAccess, including preferred_labels and idno. | |
| __filepath__ | This source takes the full server filepath from your media import directory to give you the media. Map this to ca_object_representations and use the objectRepresentationSplitter. | `{"objectRepresentationType ↪": "front", "attributes": { "media": "^__ ↪filepath__" } }` |

## RDF

This is an option for importing linked data in RDF format…

## ULAN-Linked Data

ULAN Data can be imported through an interface available in the Import menu dropdown in CollectiveAccess.

#### Omeka

Omeka data may be imported by. . .

#### WorldCat

WorldCat objects can be searched and imported using the WorldCat interface available in the Import menu dropdown. This tool uses standard import mappings to match the WorldCat source fields to fields in the CollectiveAccess profile.

These import mappings are written as described above in the xPath notation used for MARCXML.

#### CollectiveAccess

Migrating data from one CollectiveAccess installation to another can be done by setting the Source column to the appropriate ca_table.element identifier. This will map the originating data to the fields of the new installation.

### 1.31.7 Regular Expressions

#### Remove All Spaces

Changes “1982 . 30001” into “1982.30001”

```
" {
 ""applyRegularExpressions"":
 [{
      ""match"": ""(\\s)+"",
      ""replaceWith"": """"
  }]
 }"
```

### 1.31.8 Introduction

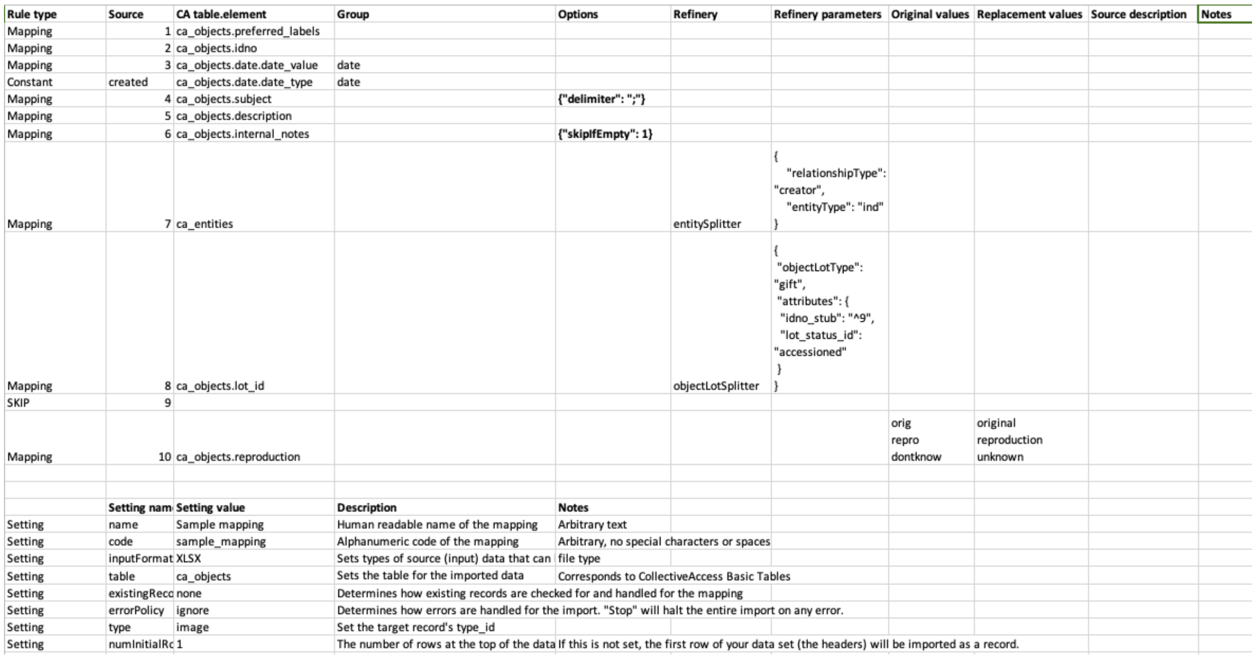
Users can map and migrate data directly from the command line or the Providence user interface (under “Import > Data”) into a Providence installation, mapping to the installation profile. An *import mapping* is a spreadsheet that defines how data is imported into CollectiveAccess. There are many settings and options in the import mapping. This documentation is organized by column, with a description of the function of each column along with the available settings for that column.

Import mappings operate under two basic assumption about your data: that each row in a data set corresponds to a single record and that each column corresponds to a single metadata element. The exception to these rules is an option called treatAsIdentifiersForMultipleRows that will explode a single row into multiple records. This is very useful if you have a data source that references common metadata shared by many pre-existing records in a single row. See the Options section for more details: http://manual.collectiveaccess.org/import/tutorial.html#options-column-5.

Running a data import involves seven basic steps:

1. Create an import mapping document (in Excel or Google Sheets) that will serve as a crosswalk between source data and the destination in CollectiveAccess.
2. Create a backup of the database by executing a data dump *before running the import* .
3. Run the import from either the command line or the graphical user interface.
4. Check the data in CollectiveAccess and look for errors or points of inconsistencies.
5. Revise your mapping accordingly.
6. Load the data dump so that the system returns to its pre-import state.
7. Run the import again.

## 1.31.9 Sample Mapping



| Rule type | Source | CA table.element | Group | Options | Refinery | Refinery parameters | Original values | Replacement values | Source description | Notes |
|---|---|---|---|---|---|---|---|---|---|---|
| Mapping | 1 | ca_objects.preferred_labels | | | | | | | | |
| Mapping | 2 | ca_objects.idno | | | | | | | | |
| Mapping | 3 | ca_objects.date.date_value | date | | | | | | | |
| Constant | created | ca_objects.date.date_type | date | | | | | | | |
| Mapping | 4 | ca_objects.subject | | **{"delimiter": ";"}** | | | | | | |
| Mapping | 5 | ca_objects.description | | | | | | | | |
| Mapping | 6 | ca_objects.internal_notes | | **{"skipIfEmpty": 1}** | | | | | | |
| Mapping | 7 | ca_entities | | | entitySplitter | {<br>"relationshipType": "creator",<br>"entityType": "ind"<br>} | | | | |
| Mapping | 8 | ca_objects.lot_id | | | objectLotSplitter | {<br>"objectLotType": "gift",<br>"attributes": {<br>"idno_stub": "^9",<br>"lot_status_id": "accessioned"<br>}<br>} | | | | |
| SKIP | 9 | | | | | | | | | |
| Mapping | 10 | ca_objects.reproduction | | | | | orig<br>repro<br>dontknow | original<br>reproduction<br>unknown | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | **Setting nam** | **Setting value** | **Description** | **Notes** | | | | | | |
| Setting | name | Sample mapping | Human readable name of the mapping | Arbitrary text | | | | | | |
| Setting | code | sample_mapping | Alphanumeric code of the mapping | Arbitrary, no special characters or spaces | | | | | | |
| Setting | inputFormat | XLSX | Sets types of source (input) data that can | file type | | | | | | |
| Setting | table | ca_objects | Sets the table for the imported data | Corresponds to CollectiveAccess Basic Tables | | | | | | |
| Setting | existingReco | none | Determines how existing records are checked for and handled for the mapping | | | | | | | |
| Setting | errorPolicy | ignore | Determines how errors are handled for the import. "Stop" will halt the entire import on any error. | | | | | | | |
| Setting | type | image | Set the target record's type_id | | | | | | | |
| Setting | numInitialRc | 1 | The number of rows at the top of the data | If this is not set, the first row of your data set (the headers) will be imported as a record. | | | | | | |

Download these files to see how the Sample Mapping applies to the Sample Data within the Sample Profile. Note that you can upload these to Google Drive and import both import mappings and source data via Google Drive.

Sample mapping.xlsx
Sample data.xlsx
Sample import profile.xml

## 1.31.10 Supported Data Input Formats

Data can be in: Exif, MODS, RDF, Vernon, FMPDSOResult, MediaBin, ResourceSpace, WordpressRSS, CSVDelimited, FMPXMLResult, MySQL, SimpleXML, WorldCat, CollectiveAccess (CA-to-CA imports), Inmagic, Omeka, TEI, iDigBio, EAD, MARC, PBCoreInst, TabDelimited, Excel, MARCXML, PastPerfectXML, ULAN

A full description of the supported import formats and how they may be referenced is available in the in the *Supported File Formats* page.

### 1.31.11 Creating a Mapping

#### Settings

Start from the sample worksheet provided above. Settings include the importer name, format of the input data, CollectiveAccess table to import to, and more. This section can be placed at the top or bottom of a mapping spreadsheet with the setting in the first column and parameter in the second. It functions separately from the main column-defined body of the import mapping.

| Setting | Description | Parameter notes | Example |
|---|---|---|---|
| name | Give your mapping a name. | Arbitrary text | My Sample mapping |
| code | Give your mapping an alphanumeric code of the mapping | Arbitrary text, with no special characters or spaces | my_sample_mapping |
| inputFormats | Sets type of source (input) data that can be handled by this import mapping. Values are format codes defined by the various DataReader plugins. | file type | XLSX |
| table | Sets the table for the imported data. If you are importing Objects, set the table to ca_objects. If you are importing Collections, set this to ca_collections, and so on. | Corresponds to the CollectiveAccess basic tables | ca_objects |
| type | Set the Type of record to set all imported records to. If you are importing Objects, what type are they? Photographs, Artifacts, Paintings, etc. This value needs to correspond to an existing value in the the types list. For objects, the list isobject_types. If the import includes a mapping to type_id, that will be privileged and the type setting will be ignored. | CollectiveAccess list item code | image |
| numInitialRowsToSkip | The number of rows at the top of the data set to skip. Use this setting to skip over column headers in spreadsheets and similar data. | numeric value | 1 |
| existingRecordPolicy | Determines how existing records in the CollectiveAccess system are checked for and handled for the mapping. Also determines how records created by the mapping are merged with other instances (idno and/or preferred label) in the data source.<br>(In CollectiveAccess, the primary ID field is "idno" and the title/name field of each record is "preferred_label".) | | none |
| | From version 1.7.9 options to skip, merge or overwrite on internal CollectiveAccess record | | |

### Rule Types (Column 1)

Each row in the mapping must have a rule defined that determines how the importer will treat the record. Available rules are:

| Rule type | Description |
|---|---|
| Mapping | Maps a data source column or table.field to a CollectiveAccess metadata element. Mappings can carry refineries (see below). |
| SKIP | Use SKIP to ignore a data source column or table.field. |
| Constant | Set a data source column or table.field to an arbitrary constant value. Include the chosen value in the Source column on the mapping spreadsheet. Matches on CollectiveAccess list item idno. |
| Rule | Performs actions during the import (such as skipping or setting data) based on conditional expressions. Made up of two parts: "Rule triggers" and "Rule actions." *TEST* See Rules page for more information: *Rules* |
| Setting | Sets preferences for the mapping (see below). |

### Source (Column 2)

The source column sets which column from the data source is to be mapped or skipped. You can also set a constant data value, rather than a mapping, by setting the rule type to "constant" and the source column as the value or list item idno from your CollectiveAccess configuration.

An explanation of the most common sources is below. A full description of the supported import formats and how they may be referenced is available in the in the *Supported File Formats* page.

| Type | Method for setting the source column |
|---|---|
| Spreadsheets | You must convert column letters to numbers. For example, if you want to map Column B of an Excel spreadsheet, you list the Source as 2. (A = 1, B = 2, C = 3, and so on.) Column B of your source data would be pulled. If on the other hand, you wish to skip this column, you would set the Rule Type to Skip and the source value to 2. |
| XML | Set the Source column to the name of the XML tag, proceeded with a forward slash (i.e. /Sponsoring_Department or /inm:ContactName) |
| XPath | XPath is a query language for selecting nodes and computing values from an XML document. It is supported for "Source" specification when importing XML. W3C offers a basic tutorial for writing XPath expressions. |
| MARC | Like other XML formats, the Source value for MARC XML fields and indicators can be expressed using XPATH. |
| FMPXML/RESULT | FileMakerPro XMLRESULT. A few things to note here due to inclusion of invalid characters in field names in certain databases (i.e. ArtBase). See Supported File Formats for rules for Source Field name rules. |

**Note:** Excel Tip: Translating A, B, C… to 1, 2, 3… can be time-consuming. Excel's preferences allow you to change columns to display numerically rather than alphabetically. Go to Excel Preferences and select "General." Click "Use R1C1 reference style." This will display the column values as numbers.

#### Special sources

A few special sources are available regardless of the format of the data being imported. These values can be useful for disambiguating the sources of data within CollectiveAccess after import.

Sometimes it's important to know, for example, which row from an Excel data set a record came from because there's not enough other data to disambiguate for testing, etc. "Special sources" addresses this by letting you map _row_ to somewhere like internal notes. To do this you include _row_ instead of a number in the source column of your mapping.

| Source | Description |
|---|---|
| `__row__` | The number of the row being imported. |
| `__source__` | The name of the file being imported. For files imported through the web interface this will be a server-side temporary filename, not the original name of the file. |
| `__filename__` | The original name of the file, when available. If the original name of the file is not available (because the uploading web browser did not report it, for instance) then the value for __source__ is returned. |
| `__filepath__` | The full path on the server to the file being imported. |
| `__now__` | The current date and time. (Available from version 1.7.9). |

### CA_table.element (Column 3)

This column declares the metadata element in the "table" set in *Settings* where the data in the source column will be mapped to in CollectiveAccess. If you are setting the source to Skip you do not need to complete this step. If you are mapping data or applying a constant value, you need to set the destination by adding the ca_table.element_code in this column.

CA_table corresponds to the CollectiveAccess basic tables, while element_code is the unique code you assigned to a metadata element in your CA configuration, or an intrinstic field in CA. For example, to map a Title column from your source data into CollectiveAccess, set the CA table.element as `ca_objects.preferred_labels`

**Mapping to Containers**

A Container is a metadata element that contains sub-elements. In order to import to specific sub-elements within a Container, you must cite the element codes for both the Container and the code for the sub-element: `ca_table.container_code.element_code`.

Example: a Date field might actually be a container with two sub-elements: a date range field for the date itself, and a date type drop-down menu to qualify the date. In this case, we would import the date from our source data as:

```
ca_objects.date.date_value
ca_objects.date.date_type
```

To map the two of these into the same container, use groups. See more in *Group (Column 4)* .

**Mapping to Related Tables**

Data will often contain references to related tables, such as related entities, related lots, related collections, related storage locations, and so on. In order to import data of one table (like ca_objects) while also creating and related records of other tables (like ca_entities), you will need to use refineries.

When your mapping includes references to a table outside the table set in the "table" Settings, you usually just need to cite the table name in this column (example: ca_entities). Then set the details in the refineries column. The exception to this is when you are creating Lot records. In this case, you set the ca_table.element_code to ca_objects.lot_id.

### Group (Column 4)

In many cases, data will map into corresponding metadata elements bundled together in a container. To continue the example above, a common container is Date, where there are actually two metadata elements - one for the date itself, and another the date's type (Date Created, Date Accessioned, etc.). Let's say in your source data there is one column that contains date values, while the next column over contains the date types.

Declaring a Group is simple. Just assign a name to each line in mapping column 4 that is to be mapped into a single container.

If Source "2" is mapping to ca_objects_date.date_value, and Source "3" is mapping to ca_objects.date.date_type, give each line the group name "Date" This will tell the mapping that these two lines are going to a single container - and won't create a whole new container for each. Any word will work, it just has to be the same for each element that goes into the container.

## Options (Column 5)

Options can be used to set a variety of conditions on the import, process data that needs clean-up, or format data with templates. This example shows some of the more commonly used options. See the complete list of options: *Mapping Options*

| Option | Description | Notes | Example |
|---|---|---|---|
| delimiter | Delimiter to split repeating values on. | | `{"delimiter": ";"}` |
| hierarchicalDelimiter | Delimiter to use when formatting hierarchical paths. This option is only supported by sources that have a notion of related data and relationship types, most notably (and for now only) the CollectiveAccess-DataReader. | | `{"hierarchicalDelimiter": "  "}` |
| formatWithTemplate | Display template to format value with prior to import. Placeholder tags may be used to incorporate data into the template. Tags always start with a "^" (caret) character. For column-based import formats like Excel and CSV the column number is used to reference data. For XML formats an XPath expression is used. While templates are tied to the specific source data element being mapped, they can reference any element in the import data set. For example, in an import from an Excel file, the template used while mapping column 2 (tag ^2 in the template) may also use tags for any other column. | There is no requirement that a template include a tag for the column being mapped. The template can reference any element in the current row, without restriction. | `{ ↪"formatWithTemplate ↪": "Column 1: ^1; ↪ Column 4: ^4"}` |

Continued on next page

Table 5 – continued from previous page

| Option | Description | Notes | Example |
|---|---|---|---|
| applyRegularExpressions | Rewrite source data using a list of Perl compatible regular expressions as supported in the PHP programming language. Each item in the list is an entry with two keys: "match" (the regular expression) and "replaceWith" (a replacement value for matches). "replaceWith" may include numbered back references in the form \n where n is the index of the regular expression parenthetical match group. | "applyWithRegularExpressions" will modify the data value being mapped for both import and comparison. Options that test values, such as "skipIfValue", will use the modified value unless _useRawValuesWhenTestingExpression_ is set. | `{`<br>`↪"applyRegularExpressions`<br>`↪": [`<br>`{`<br>`"match`<br>`↪": "([0-9]+)\\.`<br>`↪([0-9]+)",`<br>`↪"replaceWith":`<br>`↪"\\1:\\2"`<br>`},`<br>`{`<br>`"match`<br>`↪": "[^0-9:]+",`<br>`↪"replaceWith": ""`<br>`}`<br>`]`<br>`}` |
| prefix | Text to prepend to value prior to import. | From version 1.7.9 placeholder tags may be used to incorporate import data into the prefix. In previous versions, only static text was supported. | |
| suffix | Text to append to value prior to import. | From version 1.7.9 placeholder tags may be used to incorporate import data into the suffix. In previous versions, only static text was supported. | |
| default | Value to use if data source value is empty. | | |
| restrictToTypes | Restricts the the mapping to only records of the designated type. For example the Duration field is only applicable to objects of the type moving_image and not photograph. | | {"restrictToTypes": ["moving_image", "audio"]} |
| filterEmptyValues | Remove empty values from values before attempting to import. | When importing repeating values, all values are imported, even blanks. Setting this option filters out any value that is zero-length. | |

Continued on next page

Table 5 – continued from previous page

| Option | Description | Notes | Example |
|---|---|---|---|
| filterToTypes | Restricts the mapping to pull only records related with the designated types from the source. | This option is only supported by sources that have a notion of related data and types, most notably (and for now only) the CollectiveAccess-DataReader. | |
| filterToRelationshipTypes | Restricts the mapping to pull only records related with the designated relationship types from the source. | This option is only supported by sources that have a notion of related data and relationship types, most notably (and for now only) the CollectiveAccess-DataReader. | |
| skipIfEmpty | Skip the mapping If the data value being mapped is empty. | | `{"skipIfEmpty": 1}` |
| skipRowIfEmpty | Skip the current data row if the data value being mapped is empty. | | |
| skipGroupIfEmpty | Skip all mappings in the current group if the data value being mapped is empty. | | |
| skipIfValue | Skip the mapping If the data value being mapped is equal to any of the specified values. | Comparisons are case-sensitive. | {“skipIfValue”: [“alpha”, “gamma”]} |
| skipRowIfValue | Skip the current data row If the data value being mapped is equal to any of the specified values. | Comparisons are case-sensitive. | {“skipRowIfValue”: [“alpha”, “gamma”]} |
| skipGroupIfValue | Skip all mappings in the current group If the data value being mapped is equal to any of the specified values. | Comparisons are case-sensitive. | {“skipGroupIfValue”: [“alpha”, “gamma”]} |
| skipIfNotValue | Skip the mapping If the data value being mapped is not equal to any of the specified values. | Comparisons are case-sensitive. | {“skipIfNotValue”: [“beta”]} |
| skipRowIfNotValue | Skip the current data row If the data value being mapped is not equal to any of the specified values. | Comparisons are case-sensitive. | {“skipRowIfNotValue”: [“beta”]} |
| skipGroupIfNotValue | Skip all mappings in the current group If the data value being mapped is not equal to any of the specified values. | Comparisons are case-sensitive. | {“skipGroupIfNotValue”: [“beta”]} |
| skipIfExpression | Skip mapping if expression evaluates to true. All data in the current row is available for expression evaluation. By default, data is the “raw” source data. To use data rewritten by replacement values and applyRegularExpressions in your expression evaluation, set the _useRawValuesWhenTestingExpression_ to false. | | {“skipIfExpression”: “^14 =~ /kitten/”} |

Continued on next page

Table 5 – continued from previous page

| Option | Description | Notes | Example |
|---|---|---|---|
| skipRowIfExpression | Skip data row if expression evaluates to true. Data available during evaluation is subject to the same rules as in _skipIfExpression_. | | {“skipRowIfExpression”: “wc(^14) > 10”} |
| skipGroupIfExpression | Skip mappings in the current group if expression evaluates to true. Data available during evaluation is subject to the same rules as in _skipIfExpression_. | | {“skipGroupIfExpression”: “wc(^14) > 10”} |
| skipIfDataPresent | Skip mapping if data is already present in CollectiveAccess. | Available from version 1.7.9 | |
| skipIfNoReplacementValue | Skip mapping if the value does not have a replacement value defined. | Available from version 1.7.9 | |
| skipWhenEmpty | Skip mapping when any of the listed placeholder values are empty. | Available from version 1.7.9 | {“skipWhenEmpty”: [“^15”, “^16”, “^17”]} |
| skipWhenAllEmpty | Skip mapping when all of the listed placeholder values are empty. | Available from version 1.7.9 | {“skipWhenAllEmpty”: [“^15”, “^16”, “^17”]} |
| skipGroupWhenEmpty | Skip group when any of the listed placeholder values are empty. | Available from version 1.7.9 | {“skipGroupWhenAllEmpty”: [“^15”, “^16”, “^17”]} |
| skipGroupWhenAllEmpty | Skip group when all of the listed placeholder values are empty. | Available from version 1.7.9 | {“skipGroupWhenAllEmpty”: [“^15”, “^16”, “^17”]} |
| skipRowWhenEmpty | Skip row when any of the listed placeholder values are empty. | Available from version 1.7.9 | {“skipRowWhenAllEmpty”: [“^15”, “^16”, “^17”]} |
| skipRowWhenAllEmpty | Skip row when all of the listed placeholder values are empty. | Available from version 1.7.9 | {“skipRowWhenAllEmpty”: [“^15”, “^16”, “^17”]} |
| useRawValuesWhenTestingExpression | Determines whether data used during evaluation of expressions in _skipIfExpression_, _skipRowIfExpression_ and similar is raw, unaltered source data or data transformed using replacement values and/or regular expressions defined for the mapping. The default value is true – use unaltered data. Set to false to use transformed data. (Available from version 1.7.9) | Available from version 1.7.9 | {“useRawValuesWhenTestingExpression”: false} |
| maxLength | Defines maximum length of data to import. Data will be truncated to the specified length if the import value exceeds that length. | | |
| relationshipType | A relationship type to use when linking to a related record. | The relationship type code is used. This option is only used when directly mapping to a related item without the use of a splitter. | |
| convertNewlinesToHTML | Convert newline characters in text to HTML &lt;BR/&gt; tags prior to import. | | |
| collapseSpaces | Convert multiple spaces to a single space prior to import. | | |

Continued on next page

Table 5 – continued from previous page

| Option | Description | Notes | Example |
|---|---|---|---|
| useAsSingleValue | Force repeating values to be imported as a single value concatenated with the specified delimiter. | This can be useful when the value to be used as the record identifier repeats in the source data. | |
| matchOn | List indicating sequence of checks for an existing record; values of array can be "label" and "idno". Ex. ["idno", "label"] will first try to match on idno and then label if the first match fails. | This is only used when directly mapping to a related item without the use of a splitter. | |
| truncateLongLabels | Truncate preferred and non-preferred labels that exceed the maximum length to fit within system length limits. | If not set, an error will occur if overlength labels are imported. | |
| lookahead | Number of rows ahead of or behind the current row to pull the import value from. | This option allows you to pull values from rows relative to the current row. The value for this option is always an integer indicating the number of rows ahead or (positive) or behind (negative) to jump when obtaining the import value. This setting is effective only for the mapping in which it is set. | |
| useParentAsSubject | Import parent of subject instead of subject. | This option is primarily useful when you are using a hierarchy builder refinery mapped to parent_id to create the entire hierarchy (including subject) and want the bottom-most level of the hierarchy to be the subject rather than the item that is the subject of the import. | |

Continued on next page

Table 5 – continued from previous page

| Option | Description | Notes | Example |
|---|---|---|---|
| treatAsIdentifiersForMultipleRows | Explode import value on delimiter and use the resulting list of values as identifiers for multiple rows. | This option will effectively clone a given row into multiple records, each with an identifier from the exploded list. | |
| displaynameFormat | Transform label using options for formatting entity display names. Default is to use value as is. | Other options are surnameCommaForename, forenameCommaSurname, forenameSurname. See DataMigrationUtils::splitEntityName(). | |
| mediaPrefix | Path to import directory containing files references for media or file metadata attributes. | This path can be absolute or relative to the configured CollectiveAccess import directory, as defined in the app.conf _batch_media_import_root_directory_ directive. | |
| matchType | Determines how file names are compared to the match value. | Valid values are STARTS, ENDS, CONTAINS and EXACT. (Default is EXACT). | |
| matchMode | Determines whether to search on file names, enclosing directory names or both. | Valid values are DIRECTORY_NAME, FILE_AND_DIRECTORY_NAMES and FILE_NAME. (Default is FILE_NAME). | |
| errorPolicy | Determines how errors are handled for the mapping. Options are to ignore the error, stop the import when an error is encountered and to receive a prompt when the error is encountered. | Valid values are _ignore_ and _stop_. | |
| add | Always add values after existing ones even if existing record policy mandates replacement (Eg. merge_on_idno_with_replace, Etc.) | Available from version 1.7.9 | |
| replace | Always replace values, removing existing, ones even if existing record policy does not mandate replacement (Eg. is not merge_on_idno_with_replace, Etc.). | Available from version 1.7.9 | |

In the example above, multiple subject values in the same cell that are separated by semi-colons. By setting the delimiter option in the mapping, you are ensuring that these subject values get parsed and imported to discrete instances of the Subject field. Without the delimiter option, the entire string would end up a single instance of the Subject field.

### Refineries (Column 6-7)

Refineries fall into one of 5 camps: Splitters, Makers, Joiners, Getters and Builders. Each framework is designed to take a specific data format and transform it via a specific behavior as it is imported into CollectiveAccess. See the Refineries page for a complete list of refineries: *Refineries*

### Splitters

Splitter refineries can either create records, match data to existing records (following a mapping's existingRecordPolicy) or break a single string of source data into several metadata elements in CollectiveAccess. Splitters for relationships are used when several parameters are required, such as setting a record type and setting a relationship type. Using the entitySplitter, a name in a single location (i.e. column) in a data source can be parsed (into first, middle, last, prefix, suffix, et al.) within the new record. Similarly the measurementSplitter breaks up, for example, a list of dimensions into to a CollectiveAccess container of sub-elements. "Splitter" also implies that multiple data elements, delimited in a single location, can be "split" into unique records related to the imported record. See the *Splitters* page for a complete list of splitters.

### Makers

Maker refineries are used to create CollectiveAccess tour/tour stop, object lot/object and list/list item pairings. These relationships are different than other CollectiveAccess relationships for two reasons. Firstly, they don't carry relationship types. Secondly, these relationships are always single to multiple: a tour can have many tour stops, but a tour stop can never belong to more than one tour. Similarly an object can never belong to more than one lot. List items belong to one and only one list. The Maker refinery is used for these specific cases where "relationshipType" and other parameters are unnecessary.

### Joiners

In some ways Joiners are the opposite of Splitters. An entityJoiner refinery is used when two or more parts of a name (located in different areas of the data source) need to be conjoined into a single record. The dateJoiner makes a single range out of two or more elements in the data source.

### Getters

Getters are designed specifically for MYSQL data mappings. These refineries map the repeating source data through the linking table to the correct CollectiveAccess elements.

### Builders

Builders create an upper hierarchy above the to-be-imported data. Note that Splitters also create upper hierarchies with the parent parameter, but they do so above records related to the imported data. For example, let's say you were importing ca_collections and wanted to map a "Series" and "Sub-series" above imported "File" data. You'd use the collectionHierarchyBuilder refinery. However, if you were importing ca_objects and wanted to relate a "File" while building an upper hierarchy of "Series" and "Sub-series" you would use the collectionSplitter and the parent parameter. See the Builders page for a complete list of builders: *Builders*

#### Original values and Replacement values (Columns 8-9)

In some cases, particularly when you are mapping to a list element, you may need the mapping to find certain values in your source data and replace them with new values upon import. In the Original Value column, you may state all values that you wish to have replaced. Then, in the Replacement Value column, set their replacements. You can add multiple values to a single cell, so long as the replacement value matched the original value line by line. Using the Original and Replacement columns is sufficient for transforming a small range of values. But for really large transformation dictionaries, use the option "transformValuesUsingWorksheet" instead: http://manual.collectiveaccess.org/import/mappings.html#transform-values-using-worksheet.

#### Source Description & Notes (Columns 10 & 11)

These two columns are used to clarify the source and purpose of each line in the mapping and are optional. Source Description is generally a plain text label or name for the original source column to allow for easy reference to which fields are being mapped (or skipped) in the mapping. Notes provides a space to explain how and why a certain line is mapped in the manner that it is, for example, explaining why a certain value is being omitted or how an entity line is being split and related to the main record.

These fields can be useful for future reference if a mapping is intended to be used repeatedly to be sure that the selected mapping matches the source data.

### 1.31.12 Importing data

Once the import mapping is complete, you are ready to run the import. See the *Running an Import* page.

## 1.32 Basic Data Import Tutorial

### 1.32.1 Introduction

This tutorial will provide you with the very basics of writing a custom import mapping, using an example mapping document, sample data, and a basic profile configuration for Providence.

The tutorial is only intended to be a simple overview of basic import mapping principles. For more exhaustive and complete documentation of the many various options and rules, please refer to the Data Importer page.

To begin this tutorial, download the following three files.

Sample mapping.xlsx
Sample data.xlsx
Sample import profile.xml

At its simplest, the import mapping is essentially a schema crosswalk: for every data source you list, you declare a target "destination" for where the data should end up in CollectiveAccess. However, there are several additional rules and parameters you can set to ensure the "data reader" can correctly parse and process the data.

Here is a column-by-column explanation of each component of the import mapping document.

### 1.32.2 1. Rule types (Column 1)

For each line in your import mapping document, you must declare a rule type. This basically just sets what each line's job is going to be during the import. The complete list of rules can be found on the Data Importer page, but these are the rules necessary to run a basic import, and sufficient for most data imports.

| Rule type | Description |
|---|---|
| Mapping | Maps a data source (such as a column in an Excel spreadsheet or a specific tag in XML) to a CollectiveAccess metadata element. |
| Skip | Use Skip to ignore a data source. |
| Constant | Sets an arbitrary constant value. Add the value to the source column and the value will be set in the corresponding metadata element for every record that is imported. |
| Setting | Sets general preferences for the mapping overall (SEE BELOW). |

### 1.32.3 2. Settings

Every import mapping requires a few general settings, explained below. Again, the complete list of settings can be found here, but these are the most basic and required settings to get started and handle most simple imports.

In our example mapping, we have set the name to Sample Mapping and code to sample_mapping. Because our sample data is in an Excel spreadsheet, we've set the inputFormat to XLSX. Our source data contains Objects, so we've set the table to ca_objects. Our system is empty (no records have been imported yet!) so the existingRecordPolicy is set to none. Finally, we are importing photographs, which in our profile corresponds to the object type code image.

| Setting | Description | Parameter notes | Example |
|---|---|---|---|
| name | Give your mapping a name. | Arbitrary text | My Sample mapping |
| code | Give your mapping an alphanumeric code of the mapping | Arbitrary text, with no special characters or spaces | my_sample_mapping |
| inputFormats | Sets type of source (input) data that can be handled by this import mapping. Values are format codes defined by the various DataReader plugins. | file type | XLSX |
| table | Sets the table for the imported data. If you are importing Objects, set the table to ca_objects. If you are importing Collections, set this to ca_collections, and so on. | Corresponds to the CollectiveAccess basic tables | ca_objects |
| type | Set the Type of record to set all imported records to. If you are importing Objects, what type are they? Photographs, Artifacts, Paintings, etc. This value needs to correspond to an existing value in the the types list. For objects, the list isobject_types. If the import includes a mapping to type_id, that will be privileged and the type setting will be ignored. | CollectiveAccess list item code | image |
| numInitialRowsToSkip | The number of rows at the top of the data set to skip. Use this setting to skip over column headers in spreadsheets and similar data. | numeric value | 1 |
| existingRecordPolicy | Determines how existing records in the CollectiveAccess system are checked for and handled for the mapping. Also determines how records created by the mapping are merged with other instances (idno and/or preferred label) in the data source.<br>(In CollectiveAccess, the primary ID field is “idno” and the title/name field of each record is “preferred_label”.) | | none |
| | From version 1.7.9 options to skip, merge or overwrite on internal CollectiveAccess record | | |

Finally, you will notice that the source data contains a header in the first line, indicating what each column represents. Since we don't want to actually import this line as data, we'll set numInitialRowsToSkip to 1, meaning the data importer will skip the first line of the source data document.

### 1.32.4 3. Source (Column 2)

The second column in the mapping document is where you cite all data sources you wish to import. This is the first part of the crosswalk essentially. For data that is stored in Excel, citing the source is easy - simply cite the numbers of each column you want to import (Column A=1, Column B=2, and so on.)

Excel Tip: Corresponding A, B, C with 1, 2, 3 is easy enough, but when your source data has more than ten columns or so it can be kind of a pain to come up with the numeric equivalent of each letter. However, in Excel's preferences you can change the columns to display numerically rather than alphabetically. Go to Excel Preferences and select "General." Click the option to "Use R1C1 reference style." This will display the column values as numbers.

In the example we're using for this tutorial, the sample data is in Excel. However, you may need to import data that is in an XML format. XML sources are cited in xPath, which is the standard syntax for retrieving data encoded in XML. Documentation regarding xPath be found here.

Our source data sample contains 10 columns of data, and each are listed in the mapping document under Source.

Source data columns may also be referenced elsewhere in the import mapping (generally in the Options or Refinery columns described below) by prefixing the column number with a caret "^" (for example "^10"), which indicates to the mapping that the value from column 10 should be inserted.

This allows multiple columns to be combined by using the Options settings and is frequently used within the Refineries to create detailed related entities, collections etc.

### 1.32.5 4. CA table.element (Column 3)

In the mapping's third column, you declare the destination, or target, for each source.

Most of the time, the import target is simply expressed as ca_table.element code. For example, ca_objects.description in our sample mapping and profile would take the values from column 5 in the sample data and import them to the Description field in the Object editor.

The correct way to cite the primary tables can be found here. Which table you use will likely correspond (in most, but not all cases) to the table you declared in the Setting table.

When you are importing to simple free text, DateRange, Numeric, Currency, or other kinds of datatypes, ca_table.element code is about all you need.

However, there are a few cases where some additional steps are involved.

Mapping to Containers: A Container is a metadata element that contains sub-elements. In order to import to specific sub-elements within a Container, you must cite the element codes for both the Container itself, as well as the code for the sub-element that is your ultimate target.

In our sample mapping, the Date import is an example of this. In the sample profile, you'll notice that the Date field is actually a container with two sub-elements: a date range field for the date itself, and a date type drop-down menu to qualify the date.

Here, we import the date from the Column 3 in our source data to ca_objects.date.date_value, where date is the element for the container and date_value is the element code for the final import destination.

Often times when you are importing to a Container, you'll be mapping to multiple sub-elements withing the same Container instance. That's where the Group column comes in, which we'll explain in the next section.

Finally, all of the above is assuming that your data corresponds to the primary table of your import mapping. That is, you have object-level data importing to the object table.

However, data will usually contain references to related tables, such as related entities, related lots, related collections, related storage locations, and so on.

In order to import data of one table (like ca_objects) while also creating and related records of other tables (like ca_entities), you will need to use refineries, which are explained in the following sections.

But all you need to know now is that when your mapping includes references to a table outside the primary table, you usually just need to cite the table name in this column.

For example, Source 2 is mapped simply to ca_entities in the sample mapping. All of the actual details happen over in the refinery parameters.

The solo exception to this is when you are creating Lot records. In this case, you set the ca_table.element_code to ca_objects.lot_id. This exception is expressed in the sample mapping.

### 1.32.6 5. Group (Column 4)

Declaring a Group is a simple way to ensure that all of your mappings to a Container actually end up in the same Container instance. You only need to use this column when you are mapping to Container elements.

In the example, we are mapping column 3 to ca_objects.date.date_value and using the constant rule to set ca_objects.date.date_type to "date created".

But without declaring both of these distinct mapping lines members of the same Group, you'd end up with one Date container instance with the Date itself, and another Date container instance with the Date Type! To make sure both the Date itself and the date type end up in the same instance of the Date container, simply assign them to the same group in the fourth mapping column.

The name you assign the group is arbitrary, but it should be something that is recognizable to you. In our example, I've simply called the group "Date".

### 1.32.7 6. Options (Column 5)

Options, expressed in the fifth column of the mapping document, can be used to set a variety of conditions on the import, process data that needs clean-up, or format the data with templates. Our example contains just a couple of the more basic, but super useful options. A complete list of options can be found here.

| Type of Option | Description | Parameter notes | Example for "Options" column of mapping |
|---|---|---|---|
| skipIfEmpty | If the data value corresponding to this mapping is empty, skip the mapping line. | set to a non-zero value | {"skipIfEmpty": 1} |
| delimiter | Delimiter to split repeating values on. | delimiter value | {"delimiter": ";"} |

In the sample mapping, note the delimiter option set on our mapping to ca_objects.subject. Now refer to the second record in our sample data. You'll notice that there are multiple subject values in the same cell that are separated by semi-colons. By setting the delimiter option in the mapping, you are ensuring that these subject values get parsed and imported to discrete instances of the Subject field. Without the delimiter option, the entire string would end up a single instance of the Subject field.

### 1.32.8 7. Refinery (Column 6)

If your data import requires related records, then you need to use refineries. In other words, let's say in one case you are importing objects and all you need to bring in are Titles, Identifiers, Dates, and Description. No refinery will be needed here. On the other hand, say you need to import Titles, Identifiers, Dates, Description, and Creators… and the creators will be related to the Objects as Entities. This is where refineries come along.

While you can get really complex with refinery parameters, at its most basic a refinery simply creates a record, or matches on an existing record, and creates a relationship between it and the record you are importing directly from the source data.

In our example mapping, we are importing Images as ca_objects records. But using refineries, we are also creating and relating Entity records to those Object records.

Our example uses an entitySplitter, but these same principles apply to the splitters for other tables: placeSplitter, collectionSplitter, and so on.

The objectLotSplitter requires a few extra settings, all of which are cited in our example mapping.

Lastly, Splitters aren't the only type of Refinery - they're just the most common. For a complete list of refineries, go here.

### 1.32.9 8. Refinery parameters (Column 7)

In our entitySplitter example, we'll be using the most basic and commonly used refinery parameters: entityType, and relationshipType. In the objectLotSplitter we will be using another useful parameter called attributes.

| Type of Refinery parameter | Parameter notes | Example for "Refinery Parameter" column of mapping |
|---|---|---|
| relationshipType | Accepts a constant type code for the relationship type or a reference to the location in the data source where the type can be found | `{"relationshipType": "^10"}` or `{"relationshipType": "author"}` |
| entityType | Accepts a constant list item idno from the list entity_types or a reference to the location in the data source where the type can be found | `{"entityType": "individual"}` |
| attributes | Sets or maps metadata for the entity record by referencing the metadataElement code and the location in the data source where the data values can be found | `{`<br>`"attributes": {`<br>`   "biography":"^23",`<br>`   "address": {`<br>`      "address1": "^24",`<br>`      "address2": "^25",`<br>`      "city": "^26",`<br>`      "stateprovince": "^`<br>`↪27",`<br>`      "postalcode": "^28",`<br>`      "country": "^29"`<br>`   }`<br>`}`<br>`}` |

### 1.32.10 9. Original values/Replacement values (Columns 8 & 9)

In some cases, particularly when you are mapping to a list element, you may need the mapping to find certain values in your source data and replace them with new values upon import. In the Original Value column, you may state all values that you wish to have replaced. Then, in the Replacement Value column, set their replacements. You can add multiple values to a single cell, so long as the replacement value matched the original value line by line.

In our example, there is a list element called "Reproduction" with values for reproduction, original, and unknown. In our source data, however, you'll notice that the data input for these values are abbreviated (e.g "orig", "repro", and "dontknow"). By using original and replacement values, our mapping transforms "orig" to "original" and "repro" to "reproduction" so that they can match on the list item code for the corresponding values in CollectiveAccess.

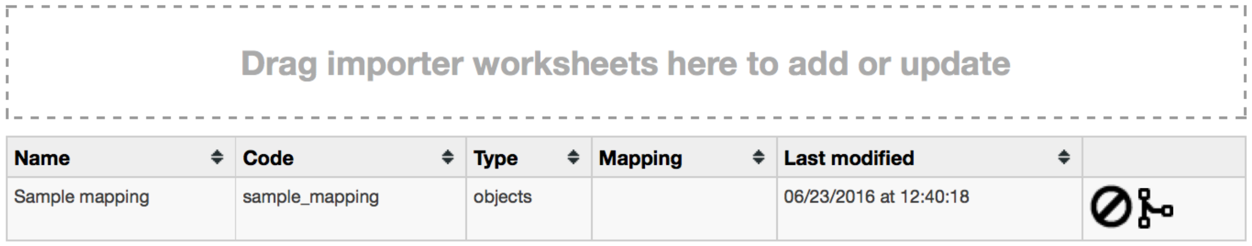
### 1.32.11 10. Source Description & Notes (Columns 10 & 11)

These two columns are used to clarify the source and purpose of each line in the mapping and are optional. Source Description is generally a plain text label or name for the original source column to allow for easy reference to which fields are being mapped (or skipped) in the mapping. Notes provides a space to explain how and why a certain line is mapped in the manner that it is, for example explaining why a certain value is being omitted or how an entity line is being split and related to the main record.

These fields can be useful for future reference if a mapping is intended to be used repeatedly to be sure that the selected mapping matches the source data.

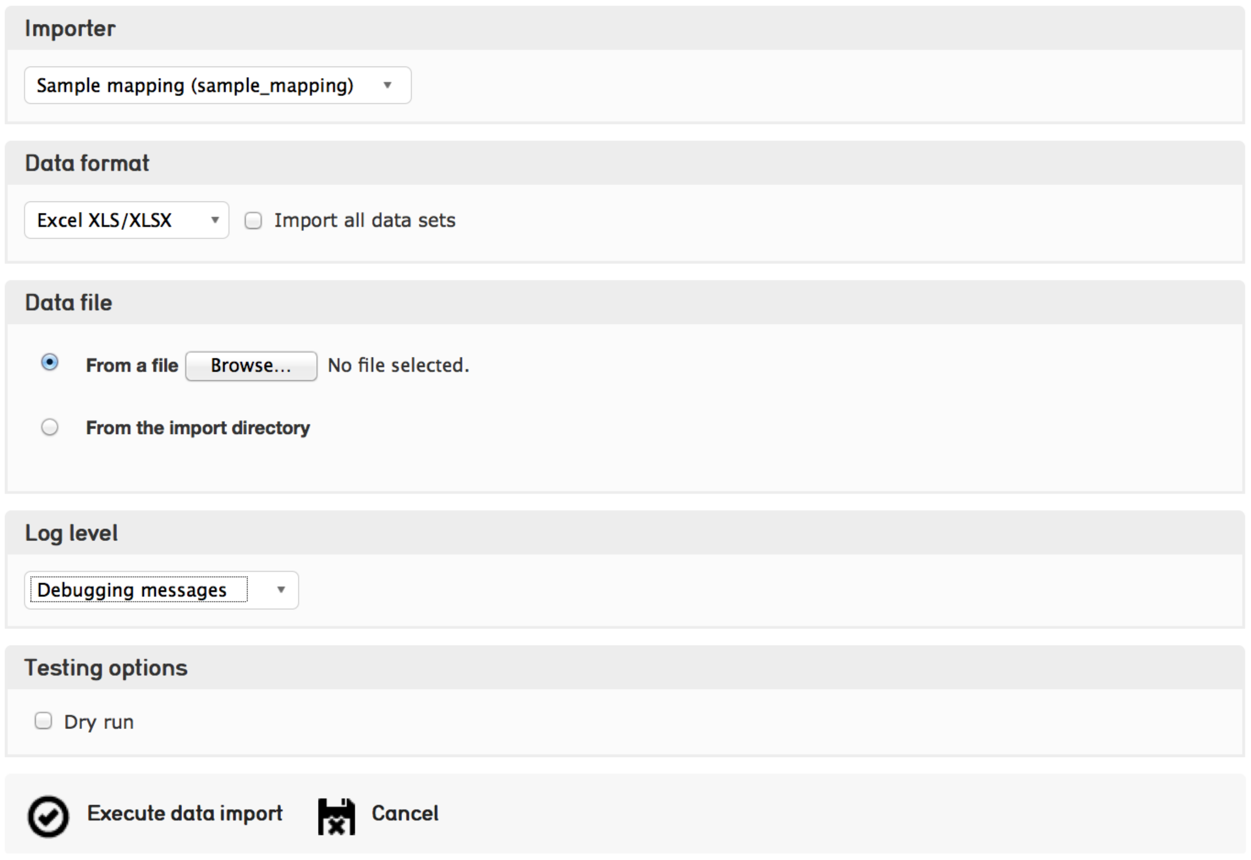
### 1.32.12 11. Importing the sample data

Once you have installed the sample profile configuration, you can load the sample mapping by navigating from the global navigtion menu to Import - Data and dragging the sample mapping file into the box labelled "Drag importer worksheets here to add or update".



| Name | Code | Type | Mapping | Last modified | |
|---|---|---|---|---|---|
| Sample mapping | sample_mapping | objects | | 06/23/2016 at 12:40:18 | |

Once the mapping is loaded, click on the icon to the right and you'll be able to upload the sample data on the following screen. From here, you can execute the data import!


Importer

Sample mapping (sample_mapping)

Data format

Excel XLS/XLSX ☐ Import all data sets

Data file

● From a file Browse... No file selected.

○ From the import directory

Log level

Debugging messages

Testing options

☐ Dry run

## 1.33 Running an Import

**Running an import from the UI**

You can execute a data import from your web browser within the Providence user interface. If you're not comfortable with using terminal, this is a good option. However, for larger imports it's recommended to use the command line, so your import is not tied up by a web browser. Remember to always back up your database before running an import, as you will likely have to tweak imports multiple times.

From the Providence navigation, go to "Import - Data." Drag and drop your import mapping XLSX to the importer list, or add the Google Drive link to your import mapping.

**Note:** Google Drive links you must have sharing settings turned on to "Any User With Link Can View". The advantage of using a Google Drive link for your mapping document is that as you tweak and edit the Google Sheet, you can update the importer by clicking the "Refresh" button in Providence. When using Excel docs, you must delete and re-upload, or change the filename to upload a new version.

Once the mapping document is loaded, you will select it to run an import on the "run import" page, where there are several settings.

| Setting | Description |
|---|---|
| Importer | This menu contains all import mappings that are loaded in the importer list. By default, this will be set to whatever mapping you chose on the previous screen. |
| Data format | This menu will automatically contain the input format value you set in the import mapping itself. If the data format does not actually correspond to the source data to be imported, change the inputFormat setting in the mapping document. |
| Data file | This is where you set the data file (or files) that are to be imported. The first option allows you to upload the data file from your machine. The second option allows you to select the file from the import directory. Use the latter option if you are importing a directory of multiple data files at once. The Third option allows you to import data from a Google Drive link, which again needs to have Share settings turned on to work. |
| Log level | This setting allows you to control the level of detail in the log. The log can capture errors, warnings, alerts, informational messages, and debugging messages. Use the latter for the most comprehensive log. |
| Testing options | Selecting "dry run" will run the import and generate a log, including errors and debugging messages, without actually creating any records in the system. It's a great way to test an import mapping without actually running an import. |

**Running an import from the terminal**

Before you begin it's a good idea to make an area for your import mappings and data that's easily accessible without an inconveniently-long file path. For the sake of this example our import material will live in a Providence directory at:

/support/project/mappings and

/support/project/data

1. *Backup your data*: Before importing, you'll maybe want to back-up your database

   ```
   mysqldump -u#name -p#password project > ~/project_date.dump
   ```

2. *Define an import*: This is done using *load-import-mapping* option of caUtils:

   ```
     cd /path_to_Providence/support
   ```

   ```
   bin/caUtils load-import-mapping --file=project/mappings/mapping1.
   xlsx
   ```

Once these commands have been run, the new mapping defined in mapping1.xlsx should have been added to imports list. If an existing mapping with the same name as defined in settings section of mapping file already existed, it has been updated.

3. *Run the import*: Once the import have been created, you'll be able to use it. For that you'll be using the utility import-data, giving the correct name to –mapping parameter.

As you'll see from:

```
bin/caUtils help import-data
```

there are several options that allow you to designate the format, data source, log preferences, etc.

To run the import:

```
bin/caUtils import-data --format=XLSX --mapping=mapping1
--source=project/data/Data.xlsx --log=project/log
```

With the PHP ncurses extension installed a display will provide moving status indicators including import progress and recent errors.

4. If something gone wrong, it's time to fix mapping or data and import again. To modify your import and rerun the utility, simply restore your database

   ```
   mysql -u#name -p#password project < ~/project_date.dump
   ```

and start the process again

## 1.34 WorldCat

## 1.35 Getty Vocabularies

## 1.36 Importing media embedded metadata

CollectiveAccess can extract and import EXIF, IPTC, XMP and technical metadata embedded in uploaded image, video, audio and document files. Import of embedded metadata can be performed on media uploaded as object representations individually in the record editor interface or in batches using the media importer.

Transformation and import of media embedded metadata employs the import system used for import of data files in various XML and delimited file formats. Crosswalks between embedded metadata extracted from media and CollectiveAccess records are specified using the same import mapping format used for general data import.

Data is extracted from media using the standard MediaInfo and ExifTool applications. When using MediaInfo, extracted metadata is presented for import as PBCore version 2.0 XML. When using ExifTool, data is presented as nested value tags. Creation of mappings for the import of output from these tools into CollectiveAccess is described in detail below.

MediaInfo is an excellent tool for extracting embedded descriptive and technical metadata from audio and video files, but returns limited metadata for images and documents. ExifTool returns comprehensive data for images, but may not capture all aspects of time-based media. Therefore you may wish to employ ExifTool for some types of media and MediaInfo for others. CollectiveAccess can be configured to automatically choose one over the other for specific file types, or to let the user decide which to use. You can also specify import mappings for specific media formats.

### 1.36.1 Installing required software

You will need to have MediaInfo and/or ExifTool installed on your server. Both applications are packaged for easy installation on most Linux distributions and the MacOS.

To install on RedHat/CentOS 7.x or 8.x:

```
yum install mediainfo exiftool
```

On Ubuntu/Debian 18.04lts:

```
apt install mediainfo exiftool
```

On MacOS (assuming you have the Homebrew package manager installed:

```
brew install mediainfo exiftool
```

Once installed, make sure the installed paths to the applications are set in your *external_applications.conf* configuration file.

### 1.36.2 The metadata extraction process

When a media file is uploaded to CollectiveAccess a sequence of processing steps are performed that result in the originally uploaded media file and a series of derivatives included in the database. Once processing is complete, extraction of embedded metadata is performed, if configured, on the originally uploaded media. Depending upon the mapping specified for the uploaded media, either MediaInfo or ExifTool are run on the uploaded media file.

When MediaInfo is used, the command-line `mediainfo` command is used with options to output extracted metadata as PBCore v2.0 XML. The command used is `mediainfo --Output=PBCore2 <filename>`, where <filename> is the path to a media file to analyze. You can use this command on the server command-line with selected files to get an idea of the format and structure of the data to be mapped.

Typical MediaInfo PBCore XML output will resemble this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated at 2020-04-20T19:43:25Z by MediaInfoLib - v19.09 -->
<pbcoreInstantiationDocument
    xsi:schemaLocation="http://www.pbcore.org/PBCore/PBCoreNamespace.html https://raw.
↪githubusercontent.com/WGBH/PBCore_2.1/master/pbcore-2.1.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.pbcore.org/PBCore/PBCoreNamespace.html">
    <instantiationIdentifier source="File Name">Self_Portrait_1969_sRGB.tif</
↪instantiationIdentifier>
    <instantiationDate dateType="file modification">2020-04-06T17:56:20Z</
↪instantiationDate>
    <instantiationDimensions unitsOfMeasure="dpi">800.000x800.000</
↪instantiationDimensions>
    <instantiationDigital>image/tiff</instantiationDigital>
    <instantiationStandard>TIFF</instantiationStandard>
    <instantiationLocation>Self_Portrait_1969_sRGB.tif</instantiationLocation>
    <instantiationMediaType>Static Image</instantiationMediaType>
    <instantiationFileSize unitsOfMeasure="byte">42993144</instantiationFileSize>
    <instantiationTracks>1</instantiationTracks>
    <instantiationEssenceTrack>
        <essenceTrackType>Image</essenceTrackType>
        <essenceTrackIdentifier source="StreamKindID (MediaInfo)">0</
↪essenceTrackIdentifier>
        <essenceTrackEncoding annotation="endianness:Big compression_mode:Lossless">
↪Raw</essenceTrackEncoding>
        <essenceTrackBitDepth>8</essenceTrackBitDepth>
        <essenceTrackFrameSize>3200x4475</essenceTrackFrameSize>
        <essenceTrackAnnotation annotationType="Title">&quot;Steve McQueen / Self-
↪Portrait, 1969 / Oil on canvas (in artist&apos;s frame) / 34 x 24 in. (86.3 x 60.9␣
↪cm) / Studio #: / Studio binder: Paintings 1969-1970  / Date of photography: /␣
↪Original photography: 4x5 Transparency&quot;</essenceTrackAnnotation>
        <essenceTrackAnnotation annotationType="ColorSpace">RGB</
↪essenceTrackAnnotation>
    </instantiationEssenceTrack>
    <instantiationAnnotation annotationType="Image_Codec_List">Raw</
↪instantiationAnnotation>
    <instantiationAnnotation annotationType="Encoded_Application_CompanyName">EPSON</
↪instantiationAnnotation>
    <instantiationAnnotation annotationType="Encoded_Application_Name">Adobe␣
↪Photoshop 21.0 (Macintosh)</instantiationAnnotation>
</pbcoreInstantiationDocument>
```

ExifTool is run with the command-line `exiftool` command and the `-json` (output in JSON format), `-g1` (group data under headings), `-a` (include all data) options. To simulate this on the server command-line use the command `exiftool -json -a -g1 <filename>` where <filename> is the path to a media file to analyze.

Typical ExifTool output with these options should resemble this example:

```
[{
  "SourceFile": "/Users/ca/Desktop/images/Self_Portrait_1969.tif",
  "ExifTool": {
    "ExifToolVersion": 11.85
  },
  "System": {
    "FileName": "Self_Portrait_1969.tif",
    "Directory": "/Users/ca/Desktop/images",
    "FileSize": "41 MB",
    "FileModifyDate": "2020:04:06 13:56:02-04:00",
    "FileAccessDate": "2020:04:06 13:56:41-04:00",
    "FileInodeChangeDate": "2020:04:06 13:56:41-04:00",
    "FilePermissions": "rw-r--r--"
  },
  "File": {
    "FileType": "TIFF",
    "FileTypeExtension": "tif",
    "MIMEType": "image/tiff",
    "ExifByteOrder": "Big-endian (Motorola, MM)",
    "CurrentIPTCDigest": "bfdbbc3492d748bae59a045d52eedeb8"
  },
  "IFD0": {
    "SubfileType": "Full-resolution Image",
    "ImageWidth": 3200,
    "ImageHeight": 4475,
    "BitsPerSample": "8 8 8",
    "Compression": "Uncompressed",
    "PhotometricInterpretation": "RGB",
    "ImageDescription": "Self-Portrait, 1969\nOil on canvas (in artist's frame)\n34 x␣
↪24 in. (86.3 x 60.9 cm)\nStudio #:\nStudio binder: Paintings 1969-1970 \nDate of␣
↪photography:\nOriginal photography: 4x5 Transparency",
    "Make": "EPSON",
    "Model": "Expression 12000XL",
    "StripOffsets": 26316,
    "Orientation": "Horizontal (normal)",
    "SamplesPerPixel": 3,
    "RowsPerStrip": 4475,
    "StripByteCounts": 42960000,
    "XResolution": 800,
    "YResolution": 800,
    "PlanarConfiguration": "Chunky",
    "ResolutionUnit": "inches",
    "Software": "Adobe Photoshop 21.0 (Macintosh)",
    "ModifyDate": "2020:04:06 12:11:15",
    "Copyright": "Permission to reproduce photography must be obtained from the Artist
↪"
  },
  "XMP-x": {
    "XMPToolkit": "Adobe XMP Core 5.6-c148 79.164036, 2019/08/13-01:06:57        "
  },
  "XMP-xmp": {
    "CreatorTool": "Adobe Photoshop 21.0 (Macintosh)",
    "MetadataDate": "2020:04:06 12:11:15-04:00",
    "CreateDate": "2020:02:05 10:46:06-05:00",
    "ModifyDate": "2020:04:06 12:11:15-04:00"
  },
```

(continues on next page)

(continued from previous page)

```
  "XMP-xmpMM": {
    "DocumentID": "adobe:docid:photoshop:da4cff7b-7f92-de48-9b5a-715bbdf53797",
    "OriginalDocumentID": "4F5F926FB3F7A36F7B9C01E4FE4BDF17",
    "InstanceID": "xmp.iid:d8d49b93-b505-47f1-ae50-1c6197730444",
    "HistoryAction": ["saved","saved","saved","saved","saved","saved","saved"],
    "HistoryInstanceID": ["xmp.iid:67850da4-0379-454a-a635-93c142bcbae3","xmp.
↪iid:77751899-131d-4f7e-a84f-f104200b29ad","xmp.iid:5a1611bc-1e40-488b-b6cd-
↪29a4dd54c2e8","xmp.iid:967f9e41-0541-4afb-9907-5a9f41452a94","xmp.iid:be011035-f7b0-
↪49d9-a712-e25b503e07f4","xmp.iid:90d2ed31-ee25-4fd5-b2a4-0ad3a1ee1b92","xmp.
↪iid:d8d49b93-b505-47f1-ae50-1c6197730444"],
    "HistoryWhen": ["2020:02:05 11:27:06-05:00","2020:02:05 11:28:12-05:00",
↪"2020:02:13 16:42:20-05:00","2020:02:13 16:53:13-05:00","2020:04:02 10:24:09-04:00",
↪"2020:04:06 12:11:15-04:00","2020:04:06 12:11:15-04:00"],
    "HistorySoftwareAgent": ["Adobe Photoshop Camera Raw 12.1","Adobe Photoshop␣
↪Camera Raw 12.1 (Macintosh)","Adobe Photoshop 21.0 (Macintosh)","Adobe Photoshop 21.
↪0 (Macintosh)","Adobe Bridge 2020 (Macintosh)","Adobe Photoshop 21.0 (Macintosh)",
↪"Adobe Photoshop 21.0 (Macintosh)"],
    "HistoryChanged": ["/metadata","/metadata","/","/","/metadata","/","/"]
  },
  "XMP-dc": {
    "Format": "image/tiff",
    "Description": "Self-Portrait, 1969\nOil on canvas (in artist's frame)\n34 x 24␣
↪in. (86.3 x 60.9 cm)\nStudio #:\nStudio binder: Paintings 1969-1970 \nDate of␣
↪photography:\nOriginal photography: 4x5 Transparency",
    "Subject": ["Painting","Self-Portrait"],
    "Title": "Self-Portrait, 1969",
    "Rights": "Permission to reproduce photography must be obtained from the Artist"
  },
  "XMP-photoshop": {
    "Credit": "© The Artist",
    "Source": "The Studio",
    "ColorMode": "RGB",
    "ICCProfileName": "Adobe RGB (1998)",
    "CaptionWriter": "Willie Mays",
    "History": "2020-04-06T12:03:13-04:00\tFile Self_Portrait_1969.tif opened\n2020-
↪04-06T12:11:15-04:00\tFile Self_Portrait_1969.tif saved\n"
  },
  "XMP-xmpRights": {
    "Marked": true
  },
  "IPTC": {
    "CodedCharacterSet": "UTF8",
    "ApplicationRecordVersion": 4,
    "Caption-Abstract": "Self-Portrait, 1969\rOil on canvas (in artist's frame)\r34 x␣
↪24 in. (86.3 x 60.9 cm)\rStudio #:\rStudio binder: Paintings 1969-1970 \rDate of␣
↪photography:\rOriginal photography: 4x5 Transparency",
    "Writer-Editor": "Willie Mays",
    "Credit": "© The Artist",
    "Source": "The Studio",
    "ObjectName": "Self-Portrait, 1969",
    "Keywords": ["Painting","Self-Portrait"],
    "CopyrightNotice": "Permission to reproduce photography must be obtained from the␣
↪Artist"
  },
  "Photoshop": {
    "IPTCDigest": "bfdbbc3492d748bae59a045d52eedeb8",
    "XResolution": 800,
```

(continues on next page)

(continued from previous page)

```
    "DisplayedUnitsX": "inches",
    "YResolution": 800,
    "DisplayedUnitsY": "inches",
    "PrintStyle": "Centered",
    "PrintPosition": "0 0",
    "PrintScale": 1,
    "GlobalAngle": 30,
    "GlobalAltitude": 30,
    "CopyrightFlag": true,
    "URL_List": [],
    "SlicesGroupName": "Self_Portrait_1969",
    "NumSlices": 1,
    "PixelAspectRatio": 1,
    "PhotoshopThumbnail": "(Binary data 3973 bytes, use -b option to extract)",
    "HasRealMergedData": "Yes",
    "WriterName": "Adobe Photoshop",
    "ReaderName": "Adobe Photoshop 2020"
  },
  "ExifIFD": {
    "ExifVersion": "0231",
    "ColorSpace": "Uncalibrated",
    "ExifImageWidth": 3200,
    "ExifImageHeight": 4475
  },
  "ICC-header": {
    "ProfileCMMType": "Adobe Systems Inc.",
    "ProfileVersion": "2.1.0",
    "ProfileClass": "Display Device Profile",
    "ColorSpaceData": "RGB ",
    "ProfileConnectionSpace": "XYZ ",
    "ProfileDateTime": "2000:08:11 19:51:59",
    "ProfileFileSignature": "acsp",
    "PrimaryPlatform": "Apple Computer Inc.",
    "CMMFlags": "Not Embedded, Independent",
    "DeviceManufacturer": "none",
    "DeviceModel": "",
    "DeviceAttributes": "Reflective, Glossy, Positive, Color",
    "RenderingIntent": "Perceptual",
    "ConnectionSpaceIlluminant": "0.9642 1 0.82491",
    "ProfileCreator": "Adobe Systems Inc.",
    "ProfileID": 0
  },
  "ICC_Profile": {
    "ProfileCopyright": "Copyright 2000 Adobe Systems Incorporated",
    "ProfileDescription": "Adobe RGB (1998)",
    "MediaWhitePoint": "0.95045 1 1.08905",
    "MediaBlackPoint": "0 0 0",
    "RedTRC": "(Binary data 14 bytes, use -b option to extract)",
    "GreenTRC": "(Binary data 14 bytes, use -b option to extract)",
    "BlueTRC": "(Binary data 14 bytes, use -b option to extract)",
    "RedMatrixColumn": "0.60974 0.31111 0.01947",
    "GreenMatrixColumn": "0.20528 0.62567 0.06087",
    "BlueMatrixColumn": "0.14919 0.06322 0.74457"
  },
  "Composite": {
    "ImageSize": "3200x4475",
    "Megapixels": 14.3
```

(continues on next page)

(continued from previous page)

```
  }
}]
```

## 1.36.3 Creating mappings

Import of media embedded metadata is managed through the same *import mapping* system used for import of stand-alone datasets. All standard options are available when performing an import of embedded metadata. Embedded imports are always performed in the context of `ca_object_representations` records, and any relationships generated will be relative to the object representation record housing the imported media.

### MediaInfo

PBCore XML data generated by MediaInfo is passed verbatim to the data importer. The required mapping is identical in format to that used for import of stand-alone PBCore v2.0 XML documents. As with all *XML-based formats* XPath is used reference to specific elements within the XML. Note that XPath expressions should omit the `pbcoreInstantiationDocument` root tag. For example, to reference the `essenceTrackType` value in the example above use `/instantiationEssenceTrack/essenceTrackType`.

Mappings for MediaInfo-based metadata extraction must include `mediainfo` in their `inputFormats` setting.

`Sample MediaInfo mapping`

### ExifTool

JSON output generated by ExifTool is converted by CollectiveAccess into a pseudo XML file using group headers (“IPTC”, “XMP-photoshop” and others in the example above) as top-level tags and sub-entries as second-level tags. For example, to reference the XMP Dublin Core description value in the example above use `/XMP-dc/Description`.

Mappings for ExifTool-based metadata extraction must include `exif` in their `inputFormats` setting.

`Sample ExifTool mapping`

Common EXIF fields and their importer source references:

| EXIF Group | EXIF Field | Importer source reference | Notes |
|---|---|---|---|
| IPTC | Caption-Abstract | /IPTC/Caption-Abstract | |
| IPTC | Writer-Editor | /IPTC/Writer-Editor | |
| IPTC | Credit | /IPTC/Credit | |
| IPTC | Source | /IPTC/Source | |
| IPTC | ObjectName | /IPTC/ObjectName | |
| IPTC | Keywords | /IPTC/Keywords | |
| IPTC | CopyrightNotice | /IPTC/CopyrightNotice | |
| XMP-dc | Format | /XMP-dc/Format | |
| XMP-dc | Description | /XMP-dc/Description | |
| XMP-dc | Subject | /XMP-dc/Subject | |
| XMP-dc | Title | /XMP-dc/Title | |
| XMP-dc | Rights | /XMP-dc/Rights | |
| ICC_Profile | ProfileDescription | /ICC_Profile/ProfileDescription | Name of color profile |
| File | FileType | /File/FileType | |
| File | MIMEType | /File/MIMEType | |
| IFD0 | ImageWidth | /IFD0/ImageWidth | |
| IFD0 | ImageHeight | /IFD0ImageHeight | |
| IFD0 | Compression | /IFD0/Compression | Compression used |
| IFD0 | Make | /IFD0/Make | Manufacturer of digitiztion device |
| IFD0 | Model | /IFD0/Model | Model of digitization device |
| IFD0 | Software | /IFD0/Software | Software used to generate file |
| IFD0 | Orientation | /IFD0/Orientation | Orientation of image, as reported by device |

### 1.36.4 CollectiveAccess configuration

User interface and logging aspects of the import process can be configured using directives in the *app.conf* configuration file.

Users can select the import mapping they wish to use at the time of upload in the editing and batch media importer interfaces when `allow_user_selection_of_embedded_metadata_extraction_mapping` is set to a non-zero value.

When allowing user selection of mappings, `allow_user_embedded_metadata_extraction_mapping_null_option` can be set to include a "no import" option. Setting this option to zero effectively forces import of embedded metadata in all cases.

If it often desirable to have CA automatically select import mappings based upon the format of the uploaded file. The `embedded_metadata_extraction_mapping_defaults` setting can be used to map media file MIME types to mappings. MIME types may be specific (Ex. image/tiff for TIFF format images) or cover entire classes using wildcards (Ex. image/* for images of any type).

```
embedded_metadata_extraction_mapping_defaults = {
    video/* = example_mediainfo_mapping,
    image/* = example_exif_tool_mapping,
    application/pdf = pdf_metadata_import
}
```

The values are the right side of the map must be valid data import mapping codes, as defined in the `code` setting of a mapping worksheet.

How much information is logged when performing an embedded metadata import can be controlled using the `embedded_metadata_extraction_mapping_log_level` setting. Valid values are DEBUG, NOTICE, INFO, WARN, ERR, CRIT and ALERT, where DEBUG logs the most (sometimes too much) information, and levels beyond ERR log only the most critical errors. It is generally best to leave this setting on DEBUG when testing and use NOTICE or INFO if DEBUG is providing too much information.

## 1.37 Export Mappings

### 1.37.1 Supported output formats

Currently: XML, MARC21, CSV

### 1.37.2 Creating a mapping

To create a mapping, first download the Excel-based export mapping template available here (File:Data Export Mapping template.xlsx). Once all of the mappings and settings have been entered into the template it can be loaded directly into CollectiveAccess (see Running an export below). The mapping is automatically checked using format-specific rules before it is added so if your mapping has any errors or ambiguities, the mapping loader will let you know.

Creating the mapping is a very dependent on the format you want to export. Specific notes and examples can be found in the section about element values and formats.

### 1.37.3 Rule types

The first column of the main mapping spreadsheet is called "Rule type". What you set here basically qualifies what this row does. Most of the rows will end up being of the "Mapping" type but there are several options available:

| Rule type | Description |
|---|---|
| Mapping | Maps a CollectiveAccess data source to a target column number, XML element/attribute or MARC field. |
| Constant | Allows you to set an element in the target format (a CSV column or an XML element/attribute to a static/constant value. If this is set, the value is taken from the 6th column in the mapping sheet ("Source"). |
| RepeatMappings | Allows repeating a list of existing mappings in a different context. If this is set, the comma-delimited list of mappings is taken from the 6th column ("Source"). See Data_Exporter#Mapping repitition. |
| Setting | Sets preferences for the mapping (see below). |
| Variable | (Available for v1.5) Allows you, using all the available features of the exporter, to assign a value to a user-defined name for later usage. See Data_Exporter#Variables. |

### 1.37.4 Hierarchical mappings

Some export formats support hierarchical relationships between mapping items. For XML this is a very core concept. To create a hierarchy, simply assign a number to a mapping in the 2nd column of the Mapping sheet and then reference that number in other rows (i.e. for other items) in the 3rd row, which is typically named "Parent ID". The second item will then become a direct child if the first one. In theory, those hierarchies can be nested very deep but in practice the format implementations may apply restrictions.

## 1.37.5 Source

The value for the 5th column in the mapping sheet can be any CollectiveAccess bundle specifier. See API:Getting_Data#Bundle_specifiers for details. This usually specifies the actual data that is pulled into this item. Can be set to arbitrary text for items with static content or be left empty for items without content (e.g. wrapping elements in XML or empty columns in CSV).

Note that if the context for the current mapping is changed, there are a couple of special keys available for the source column. For more information see the description for the "context" option in the table below.

## 1.37.6 Element values and general notes on specific formats

The 4th column of the mapping sheet is named 'Element'. This is a very format-specific setting where you enter the name of the element you want to put your field data in. See below for a description of the formats.

## 1.37.7 XML Element values

The XML format implementation allows valid XML element names as values for the "Element" column. If you want to specify an XML attribute, prefix the name with an @. The attribute will then be appended to the hierarchy parent (which can't be another attribute). The mapping item hierarchy pretty much represents the XML tree that will be constructed from it.

Say you have the following very simple part of a mapping sheet and you export a single object.

| Rule type | ID | Parent ID | Element | Source | Options |
|---|---|---|---|---|---|
| Mapping | 1 | | object | | |
| Mapping | 2 | 1 | @idno | ca_objects.idno | |
| Mapping | 3 | 1 | title | ca_objects.preferred_labels | |

What you end up with as export for a given objects is something like the following:

```
<object idno="00001">
   <title>My very cool object</title>
</object>
```

## 1.37.8 MARC Element values

Let's start off by saying that MARC is a very old and very specific format. Creating MARC mappings can be a bit painful. Make yourself familiar with the format before you dive into the following description.

In MARC mappings, the Element value is either a control field or a data field definition. For control field definitions, simply enter the field code (like '001') here. For data field definitions, enter the field code, followed by a forward slash and both indicator characters. For details on valid field codes and indicators, please refer to the MARC documentation. For empty/unused indicators, use the pound sign (#). Valid examples are 001 300/## 490/1#

Mapping items with data field definitions also shouldn't have any source definition or static data. The data resides in subfields, which should be separate mapping items with a hierarchical relationship (via Parent ID) to the field definition. For instance, you'd define an item for the data field "300/##". Suppose it had the ID 1. This field (like every data field) has a couple of subfields [1], namely a through g and 3, 6, 8 (leave out the $ character from the original documentation). Now create separate mapping items for each subfield you need, pull in the CA data you want using the 'Source' field in the mapping sheet and fill in the Parent ID "1", the identifier of the data field. Here's an example in table form (which may not make sense from a MARC standpoint but we're only trying to explain the format here, not the semantics of MARC fields):

| Rule type | ID | Parent ID | Element | Source | Options |
|---|---|---|---|---|---|
| Mapping | 1 | | 1 | ca_objects.idno | |
| Mapping | 2 | | 300/## | | |
| Mapping | 3 | 2 | b | ca_objects.preferred_labels | |

An example export for a single object looks like this then. Note that we selected the 'readable' format for the MARC exporter, more info on format-specific settings are below.

```
LDR
001     00001
300 ## _bMy very cool object
```

## 1.37.9 Variables

This feature allows you, using all the available features of the exporter, to assign a value to a user-defined identifier for later usage. The value can be anything you can pull from the database. The ''identifier'' should ''only contain alphanumeric text, dashes and underscores''. Otherwise the mapping spreadsheet will fail to load. For example: type, my_variable, some-value, somethingCamelCase.

The identifier (essentially the name) that you assign to the variable goes into the element column. Since variable don't end up in the export, this column has no other use. Below is a simple example.

The main (and for the moment only) use for variables are conditional mappings. Say you have two objects, a document and a photo. And say you have an attribute 'secret_info' that is valid for both object types but that you only want to have in your export for photos. You could build two different mappings for these cases or you could use a variable to assign the object type to a user-defined identifier and then use the skipIfExpression option for the mapping in question.

A good way to think of variables is that they are mappings that don't end up in the actual export. They respect the current context, the current place in the hierarchy, everything.

| Rule type | ID | Parent ID | Element | Source | Options |
|---|---|---|---|---|---|
| Variable | | | type | ca_objects.type_id | |
| Mapping | 1 | | object | | |
| Mapping | 2 | 1 | @idno | ca_objects.idno | |
| Mapping | 3 | 1 | secret | ca_objects.top_secret | { "skipIfExpression" : "^type!~/49/" } |

We use the "type" variable in the skipIfExpression setting for the top_secret mapping. For more info on this setting, see the setting description below.

## 1.37.10 Settings

These are configuration options that apply to the whole exporter mapping.

| Setting | Description | Parameter notes | Example |
|---|---|---|---|
| exporter_format | Sets the format used for this mapping. | Restricted list, at the moment 'XML', 'MARC' and 'CSV' are supported. | XML |
| code | Alphanumeric code of the mapping | Arbitrary, no special characters or spaces | my_mapping |
| name | Human readable name of the mapping | Arbitrary text | My mapping |
| table | Sets the table for the exported data | Corresponds to CollectiveAccess Basic Tables | ca_objects |
| wrap_before | If this exporter is used for an item set export (as opposed to a single item), the text set here will be inserted before the first item. This can for instance be used to wrap a repeating set of XML elements in a single global element. The text should be valid for the current exporter format. | Arbitrary string value | <rdf:RDF xmlns:dc="http://purl.org/dc/elements/1 ...> |
| wrap_after | If this exporter is used for an item set export (as opposed to a single item), the text set here will be inserted after the last item. This can for instance be used to wrap a repeating set of XML elements in a single global element. The text has to be valid for the current exporter format. | Arbitrary string value | </rdf:RDF> |
| wrap_before_record | Same as wrap_before but only applies to single item/record exports. | Arbitrary string value | <mySingleRecordWrap xml:id="fooBar"> |
| wrap_after_record | Same as wrap_after but only applies to single item/record exports. | Arbitrary string value | </mySingleRecordWrap> |
| typeRestrictions | If set, this mapping will only be available for these types. Multiple types are separated by commas or semicolons. Note that this doesn't work very well for batch exports because search results or sets typically consist of records of multiple types. The exporter select dropdown always shows all exporters for that table, but when you actually run the export in batch mode, it will filter according to the restriction, which can get a little confusing when you look at the result. | comma- or semi-colon separated list of valid type codes for this table | image,document |
| MARC_outputFormat | MARC supports a couple | readable', 'raw' or 'xml'. readable refers | xml |

### 1.37.11 Options

Each mapping item (i.e. a line in the mapping spreadsheet) can have its own settings as well. To set these settings, you can fill out the 6th column of the mapping sheet, called 'Options'. The options must be filled in in JavaScript Object Notation. If you set this value and it's not formatted properly, the mapping loading tool will throw an error. Here's a description of the available options:

| Options | Description | Parameter notes | Example |
|---|---|---|---|
| default | Value to use if data source value is blank | Arbitrary text | "No value" |
| delimiter | Delimiter to used to concatenate repeating values | Usually a single character like a comma or semi-colon | |
| prefix | Text to prepend to value prior to export | Arbitrary text | Dimensions are: |
| suffix | Text to append to value prior to export | Arbitrary text | feet |
| template | Format exported value with provided template. Template may include caret (^) prefixed placeholders that refer to data source values. | See the [Bundle_Display_Templates] article for details on the syntax | ^height |
| maxLength | Truncate to specified length if value exceeds that length | Integer | 80 |
| repeat_element_for_multiple_values | Some source values may select multiple values, for instance for repeatable metadata elements. If this is the case and this option is set, the current mapping item is repeated for each value instead of them being put into a single string using the delimiter option | 1 or 0. defaults to 0 | 1 |
| filterByRegExp | Allows filtering values by regular expression. Any value that does NOT match this PCRE regular expression is filtered and not exported | Insert expression without delimiters. Has to be valid expression. | [A-Za-z0-9]+ |
| locale | Locale code to use to get the field values from the database. If not set, the system/user default is used. | Valid locale code | de_DE |
| context | **This is used to switch the context for this mapping item (and all hierarchy children) to a different record (usu** It is also possible to switch context to an attribute of the current record. This helps properly process repeatable containers as encapsuled sub-exports. If the context is switched to a container like ca_entities.address, all elements of the container are available in the source column for all child mappings. They are addressed by | Either a related table name like 'ca_entities', a metadata element bundle specifier (ca_entities.address) or one of the literals 'children' or 'parent' for hierarchy traversal. | ca_entities |

Below is a properly formatted example in JSON that uses some of these options:

```
{
    "default" : "No value",
    "delimiter" : ";",
    "maxLength" : 80,
    "filterByRegExp" : "[A-Z]+"
}
```

### 1.37.12 Processing order

In some cases the order in which the options and replacements (see next sub-section) are applied to each value can make a significant difference so it's important to note it here:

1) skipIfExpression (available for v1.5)
2) filterByRegExp
3) Replacements (see below)
    a) If value is empty, respect 'default' setting
    b) If value is not empty, use prefix and suffix
5) Truncate if result is longer than maxLength

### 1.37.13 Replacements

While looking at the exporter mapping template you might have noticed that there's a second sheet called 'Replacements' in there. This can be used to assign replacements to each mapping item. The first column references the ID you set in the 2nd column of the mapping item table. The second column defines what is to be replaced. This again should be a PCRE-compatible regular expression without delimiters. The 3rd column defines what value should be inserted for the matched values. These conditions are applied to each matching value in the order they've been defined, i.e. if you have multiple replacements for the same mapping item, the incoming value is first passed through the first replacement, the result of this action is then passed in to the second replacement, and so on . . .

[Useful note for advanced users and PHP programmers]

The values are passed through preg_replace, the 'pattern' being the 2nd column value (plus delimiters) and the 'replacement' being the value from the 3rd column. This allows you to do pretty nifty stuff, for instance rewriting dates:

Search column: (w+) (d+), (d+) Replace column: $2 $1 $3 value: April 15, 2003 result: 15 April 2003

### 1.37.14 Mapping repitition

The 'RepeatMappings' rule type allows you to repeat a set list of mappings in a different context without actually defining them again. This is, for instance, very useful when creating EAD exports of hierarchical data where the basic structure is always the same (for archdesc, c01, c02, etc.) but the context changes. It's basically a shortcut that saves a lot of work in certain scenarios. Note that all hierarchy children of the listed items are repeated as well.

If you create a RepeatMappings rule, the mapping loader expects a comma-delimited list of references to the 2nd column in the Mapping sheet. It also really only makes sense to create this type of rule if you change the context in the same step. A simple example could look like this:

| Rule type | ID | Parent ID | Element | Source | Options |
|---|---|---|---|---|---|
| Mapping | 1 | | root | | |
| Mapping | 2 | 1 | label | ca_objects.preferred_labels | |
| Mapping | 3 | 1 | idno | ca_objects.idno | |
| Mapping | 4 | 1 | children | | |
| RepeatMappings | | 4 | child | 2,3 | { "context" : "children" } |

In this case, the 'child' element would be repeated for each hierarchy child of the exported item because of the context switch and for each of those children, the exporter would add the label and idno elements.

### 1.37.15 Running an export

The export can be executed through caUtils. To see all utilities ask for help after cd-ing into support

cd /path_to_Providence/support bin/caUtils help

To get further details about the load-export-mapping utility:

bin/caUtils help load-export-mapping

To load the mapping:

bin/caUtils load-export-mapping –file=~/my_export_mapping.xlsx

Next you'll be using the utility export-data. First, have a look at the help for the command to get familiar with the available options.

bin/caUtils help export-data

Essentially there are 3 export modes:

### 1.37.16 1) Export a single record

Since the scope of a mapping is usually a single record, it's easy to use a mapping to export a record by its identifier. Suppose you have a ca_objects XML mapping with the code 'my_mapping'. To use this to export the ca_objects record with the primary key identifier (not the custom idno!) 550 to a new file ~/export.xml, you'd run this command:

bin/caUtils export-data -m my_mapping -i 550 -f ~/export.xml

### 1.37.17 2) Export a set of records found by custom search expression

In most real-world export projects you'll need to export a set of records or even all your records into a single file. The exporter utility allows this by letting you specify a search expression with the -s parameter that selects the set of records used for export. The records are simply exported sequentially in the order returned by the search engine. This sequence is wrapped in the wrap_before and wrap_after settings of the exporter, if set. If you want to export all your records, simply search for "*". This example exports all publicly accessible files to a file ~/export.xml:

bin/caUtils export-data -m my_mapping -s "access:1" -f ~/export.xml

### 1.37.18 3) Export a diverse set of records ("RDF mode")

[For advanced users] The error handling in this portion of the code is very poor so you're pretty much left on an island if something goes wrong.

Sometimes a limited export scope to for example ca_objects like in the previous example is not enough to meet the target format requirements. Occasionally you may want to build a kind of 'mixed' export where records from multiple database entities (objects, list items, places, . . . ) are treated equally. We have found this requirement when trying to use the exporter to generate an RDF graph, hence the name. The export framework originally wasn't designed for this case but the caUtils export-data command offers a way around that. The switch –rdf enables this so called "RDF mode". In this mode, you again use -f to specify the output file and you have to provide an additional configuration file (see Configuration_File_Syntax) which tells the exporter about the records and corresponding mappings which will be used for this export.

Here is a minimal example that uses all the available features:

`wrap_before = ""` `wrap_after = ""`

```
nodes = {
   my_images = {
      mapping = object_mapping,
         restrictBySearch = "access:1",
         related = {
            concepts = {
               restrictToRelationshipTypes = [depicts],
               mapping = concept_mapping,
            },
            agents = {
               restrictToTypes = [person],
               mapping = agent_mapping,
            },
        }
    },
}
```

While processing this configuration, the exporter essentially builds one big list of records and corresponding mappings to export. There are no duplicates in this list, if object_id 23 is selected by two different node type definitions or by multiple related definitions, it is still only exported once, using the mapping provided by the first definition.

Here is an example of how to run an RDF mode export:

```
bin/caUtils export-data --rdf -c ~/rdf_mode.conf ~/export.xml
```

## 1.37.19 RDF Mode configuration file options

| Setting | Description |
|---|---|
| wrap_before | Text to prepend before the export. |
| wrap_after | Text to append after the export. |
| nodes | List of primary node type definitions to be used for this export |

## 1.37.20 Node type definition options

| Setting | Description |
|---|---|
| mapping | Mapping to be used for this type of node. Has to be an existing mapping code. |
| restrictbySearch | Restrict exported records using a search expression |
| related | List of related records also to be included in the global node set. You can use this for example to make sure you only export list_items that are actually actively used as vocabulary terms for objects, meaning you don't have to create an extra node type (which would potentially export all list items in your database) for this. |

### 1.37.21 'related' options

| Setting | Description |
|---|---|
| mapping | Mapping to be used for this type of related item. Has to be an existing mapping code. |
| restrictToRelationshipTypes | Restrict selected related records by relationship types. Has to be a list or empty. |
| restrictToTypes | Restrict selected related records by record types (e.g. entity type). Has to be a list or empty. |

### 1.37.22 Misc Setting and Options

#### Exporting values from Information Services (e.g Library of Congress, Getty)

If your CollectiveAccess configuration includes information services, such as Library Of Congress Subject Headings or Getty's Art and Architecture Thesaurus, you can export these in the exact same way as you would export other kinds of metadata elements.

However, in order to comply with certain XML formats (like MODS of TEI) you may find that you need to extract the terms' URI and export these to an attribute while exporting the label name to an element.

To grab an information service term's URI, you can simply append ".uri" or ".url" to the Source.

For example, if your Getty AAT element happens to be called "ca_objects.aat" and you wish to export the URI, simply express the source as "ca_objects.aat.uri". This will give you the URI while the simple "ca_objects.aat" will get you the label name as before.

LC services work a little differently. For these, you must append to the source ".text" to get the label name and ".id" to get the URI.

For example:

`ca_objects.lcsh_terms.text` will get you the label name of all lcsh terms on the record. `ca_objects.lcsh_terms.id` will get you the URI for these terms.

## 1.38 OAI-PMH Provider

CollectiveAccess features an implementation of the OAI Protocol for Metadata Harvesting that serves database records using virtually arbitrary XML-based formats. It supports all requests defined in the protocol and is made possible through the Data_Exporter framework.

In order to make your data available via OAI-PMH you first have to create at least one XML mapping for the data exporter. Data providers are required to provide at least Dublin Core so we suggest you start with that. You can, however, provide any number of formats so long as you create the corresponding export mappings in CollectiveAccess.

For now we assume that you created a mapping with the code 'oai_dc' that describes how your data is mapped to Dublin Core XML.

### 1.38.1 oai_provider.conf configuration

This is the core configuration file for this feature. If you want to provide only one the Dublin Core mapping you just created, you're pretty much good to go with the stock configuration. Just fill in the mapping code. The interesting part of the configuration is under the 'providers' key. Here you can define an arbitrary number of endpoints where harvesters can go and gather your data. You could, for instance, define 1 endpoint for your object/item records and one for your collections. Each of those is a full-fledged standalone OAI-PMH provider accessible via the following URL pattern:

```
http://www.mydomain.org/service.php/OAI/<provider_key>
```

In the default config there's only one provider with the code 'dc'. It can be accessed like so:

```
http://www.mydomain.org/service.php/OAI/dc
```

Within each provider configuration you have a number of available settings. They are described in the table below.

| Setting | Description | Example value |
|---|---|---|
| name | Used as repositoryName for the response to the Identify verb. By default it is imported from your setup.php. | My repository |
| admin_email | Used as adminEmail for the response to the Identify verb. By default it is imported from your setup.php | me@mydomain.org |
| setFacet | A browse.conf facet code used to divide the data set you're providing into so called 'sets' (OAI terminology, not necessarily equivalent to CollectiveAccess sets) and build the response to the ListSets verb. By default associated collections are used. | collection_facet |
| query | Search query that allows you to impose arbitrary restrictions on the record set that is being provided. By default all records ('*') are served. | ca_objects.idno:0001* |
| formats | List of the metadata formats that are available for this provider. Further information on the format definition is below. | see below |
| default_format | Code of the format to use if the metadataPrefix argument is omitted, i.e. if the harvester doesn't specify which format he wants. You should almost always serve basic Dublin Core in those cases. | oai_dc |
| identiferNamespace | Namespace to use to build globally unique identifiers from your local CollectiveAccess record ids. The identifiers look like this: oai:<namespace>:<localID> | whirl-i-gig.com |
| dont_enforce_access_settings | if set, no access checks are performed | 0 |
| public_access_settings | list of values for 'access' field in objects, entities, places, etc. that allow public (unrestricted) viewing | [1] |
| privileged_access_settings | list of values for 'access' field in objects, entities, places, etc. that allow privileged viewing (ie. user in on a privileged network as defined below) | [1,2] |
| privileged_networks | List of IP address to consider "privileged" (can see items where access = 1 or 2) It is ok to use wildcards ("*") for portions of the address to create class C or B addresses, e.g. 192.168.1.5, 192.168.1.* and 192.168.*.* are all valid and increasingly broad | [192.168.6.*] |
| dont_cache | Determines if search or browse results used to built responses are cached or not | 1 |
| show_deleted | Determines if deleted records are included in list responses | 0 |

### 1.38.2 Format Definition

The definition of a single format used by a provider configures the exporter mapping that should be used for this format as well as some metadata about the format itself. The type of records served (objects, entities, . . . ) is defined by the exporter mapping. We strongly recommend not to mix mappings for different types in one provider. If you want to provide, say, your objects and your collections, you should configure 2 separate providers to do this, otherwise harvesters could get inconsistent results for the same identifiers.

The key of the format definition is the so called metadataPrefix. It is used to address these formats in OAI-PMH requests (and also for the 'default_format' setting).

| Setting | Description | Example value |
|---|---|---|
| mapping | Code of the exporter mapping to use for this provided format | oai_dc |
| schema | Used only to describe this format for the ListMetadataFormats verb. If you want schema definitions to appear in the protocol responses, they should be part of your export mapping. | http://www.openarchives.org/OAI/2.0/oai_dc.xsd |
| metadataNamespace | Used only to describe this format for the ListMetadataFormats verb. If you want namespace definitions to appear in the responses, they should be part of your export mapping. | http://www.openarchives.org/OAI/2.0/oai_dc/ |

An example configuration with only one format (metadataPrefix: oai_dc) served by the provider could look like this:

```
        formats = {
        oai_dc = {
                mapping = ca_objects_oai_dc,
                schema = http://www.openarchives.org/OAI/2.0/oai_dc.xsd,
                metadataNamespace = http://www.openarchives.org/OAI/2.0/oai_dc/,
        }
},
```

### 1.38.3 Testing your setup

To explore your collection using the OAI-PMH provider you just set up, you can for instance use the OAI Repository Explorer maintained by the University of Cape Town. This tool obviously only works if your provider is accessible online. You can also test the results by accessing provider via the following:

```
https://mydomain.com/service.php/OAI/dc?verb=ListRecords&metadataPrefix=oai_dc
```

Be sure to change out the metadataPrefix if it something other than oai_dc.

### 1.38.4 DPLA

Partner hubs of the Digital Public Library of America can provide metadata to the DPLA by setting up an OAI-PMH provider servicing data in either DublinCore, MARC, or MODS. Consult with the DPLA for other supported formats, or refer to the DPLA metadata specification crosswalk here.

## 1.39 External Export Framework

**Note:** This feature is available from CollectiveAccess version 1.7.9.

CollectiveAccess can interact with other external systems, including digital preservation and data backup platforms, using the external export framework. The framework provides a pipeline for assembly, packaging and transmission of CollectiveAccess-managed metadata and media to other applications. A variety of standard formats and protocols are supported and may be mixed and matched to facilitate interoperation.

The framework operates at the record level, creating packages incorporating metadata, media (images, audio, video, documents) and documentation such as checksums and use statements for individual objects, collections and other record types. Record metadata may be generated in any format supported by the *metadata export system*, including

XML, tab, CSV and MARC, and can include metadata from related records. Multiple metadata exports may be included in a single package, as well as any available versions of associated media.

Package formats include interchange standards such as BagIT and widely used container formats such as ZIP. Once packages are created, the framework can transfer them to external systems using protocols such as Secure FTP.

The framework is designed to be extensible. It is possible to add support for additional package formats or data transport protocols by creating software plugins.

## 1.39.1 Configuration

Behavior of the external export framework is controlled using *targets*. A target is a set of configuration defining what data is to be exported, how that data is to be packaged, what criteria trigger creation of a package and where packages are ultimately sent. You can configure as many targets as required, each with its own packaging, destinations and other characteristics.

Targets are defined in the *external_exports.conf* configuration file. Each target has a unique code and a dictionary of configuration values. Within the dictionary, a few top-level settings define basic parameters:

| Setting | Description | Example value |
|---|---|---|
| label | The name of the target, for display. | CTDA MODS export |
| enabled | Indicates if the target should be processed or not. Set to 0 to disable the target or 1 to enable. | 1 |
| table | Defines the table this target exports records from. | ca_objects |
| restrictToTypes | An optional list of one or more types for the export *table*. If set, only records with the specified types will be exported. | [books, documents] |
| checkAccess | An optional list of one or more access values to filter record on. If set, only records with the specified access values will be exported | [1] (export only records with an access status of “1”, which is typically used to indicate the record is public) |

Detailed configuration is contained in three blocks:

- *triggers* defines the criteria that will trigger export of a record to this target.
- *output* defines the data to be exported

```
targets = {
    mods_export_to_sftp_server = {
        label = MODS export,
        enabled = 1,

        table = ca_objects,
        checkAccess = [1],

        triggers = {
            lastModified = {
                from_log_timestamp = 4/1/2020,
                #from_log_id = 20000,
                #query = cooking
            }
        },

        output = {
```

(continues on next page)

(continued from previous page)

```
            format = BagIT,
            name = "EXPORT_^ca_objects.idno",
            content = {
                mods.xml = {
                    type = export,
                    exporter = mods_exporter_with_guid
                },
                . = {
                    type = file,
                    files = {
                        ca_object_representations.media.original = {
                            delimiter = .,
                            components = {^original_filename }
                        }
                    }
                }
            },
            options = {
                file_list_template = "^ca_objects.idno, ^filename, ^filesize_for_
↪display, ^mimetype",
                file_list_delimiter = ";"
            }
        },

        destination = {
            type = sFTP,
            hostname = my-sftp-server@example.net,
            user = my_user,
            password = my_password,
            path = "path/to/where/packages/are/uploaded"
        }
    }
}
```

### 1.39.2 Running an export

How to run an export here.

### 1.39.3 Extending the framework

Overview of plugin system here

## 1.40 User Access Control

## 1.41 Maintenance Functions

## 1.42 Command-line utilities

Info about caUtils to come

## 1.43 Settings

### 1.43.1 User interface settings

#### Relationship bundles

Relationship bundles provide a user interface for managing relationships between records. A range of functionality is provided and can be controlled using the settings described below. Many settings are shared across all relationship bundles, but some are only available for specific bundles.

#### Options available for all relationship bundles

TODO: VERIFY THIS IS COMPLETE

| Setting | Description | Valid values | Version notes |
|---|---|---|---|
| restrict_to_relationship_types | Comma separated list relationship type codes to limit display of related records to. | One or more relationship type codes defined for the relevant relationship, separated by commas. Leave blank to display all related records, regardless of relationship type. | |
| restrict_to_types | Comma separated list of related record types to limit display of related records to. | One or more related record type codes, separated by commas. Leave blank to display all related records regardless of type. | |
| dont_include_subtypes_in_type_restriction | Controls whether type restrictions are automatically expanded to include sub-types. Default default expansion is performed. Set to a non-zero value to prevent expansion. | 0,1 | |
| display_template | A *display template* used to format for display metadata from the related representation. The template is evaluated relative to the relationship (Eg. ca_objects_x_entities for object-entity relationships) | | |
| allowedSorts | Defines which bundles may be used to interactively sort the list of related records. Any valid record or relationship intrinsic or metadata element may be specified. Separate multiple bundles with commas. Values specified here will be included in the sort menu for the bundle. | | |
| disableSorts | Controls whether sorting controls for related records are displayed. Set to a non-zero value to disable sorting. By default sorting controls are displayed. | 0,1 | |
| sort | Bundle to use to sort the list of related records on initial load. Omit to use the natural sort order a specified by the user via drag-and-drop. | Any valid sortable intrinsic or metadata element bundle. | |
| sortDirection | The direction of the sort on initial load. Use ASC for ascending and DESC or descending. Default is ASC. | ASC or DESC | |
| showCount | Controls whether the count of related records is shown in the bundle title bar. Default is 0 (no count). Set to a non-zero value to display the count. | 0,1 | Available as of version 1.7.9 |
| dontShowDeleteButton | Controls whether a delete button is show for each related record. Default is 0 (show delete button). Set to a non-zero value to remove delete buttons. | 0,1 | |
| minRelationshipsPerRow | Minimum number of related records. If set to a non-zero value it will not be possible to delete relationships once the minimum is reached. If set to zero, or omitted, no minumum is enforced. Default is 0. | Any integer >= 0 | |
| maxRelationshipsPerRow | Maximum number of related records. If set to a non-zero value it will not be possible | Any integer >= 0 | |

### Bundle: ca_object_representations

The `ca_object_representations` bundle provides the primary interface for associating uploaded media representations with other records. For all CollectiveAccess versions functionality includes upload and preview of individual media, limited editing and display of representation metadata, drag and drop ordering, download and more.

As of CollectiveAccess version 1.7.9 an expanded interface is available that offers batch upload of media files, greatly expanded metadata editing and display and improved incremental loading and performance. Both the old "CLASSIC" interface and the new expanded interface ("NEW_UI") are supported in version 1.7.9. In future versions support for the "CLASSIC" interface may be dropped.

| Setting | Description | Valid values | Version notes |
|---|---|---|---|
| restrict_to_relationship_types | Comma separated list relationship type codes to limit display of related representations to. This setting is not relevant when displaying representations directly related to objects, as object-representation relationships do not support relationship types. | One or more relationship type codes defined for the relevant relationship, separated by commas. Leave blank to display all related representations, regardless of relationship type. | |
| display_template | A *display template* used to format for display metadata from the related representation. The template is evauated relative to the representation relationship (Eg. ca_objects_x_object_representations for object-representation relationships) | | As of version 1.7.9 additional template tags are available, providing a range of preformatted information for representation media. See the "special placeholders" section in *display template* for a list of tags. |
| uiStyle | Enables the new (as of version 1.7.9) representation bundle, which offers batch upload and in-bundle representation metadata editing. Set to "CLASSIC" for the pre-1.7.9 bundle format, or "NEW_UI" for the new bundle. The default is "CLASSIC" | CLASSIC or NEW_UI | Available as of version 1.7.9 |
| showBundlesForEditing | Selected intrinsics and metadata elements to allow editing on. Separate multiple bundles with commas. Bundles will be displayed for editing in a fixed order regardless of the order specified here. If a specific order of bundles is needed in the editing form, use the "showBundlesForEditingOrder" setting to order the bundles listed here. | Any valid sortable intrinsic or metadata element bundle. | Available as of version 1.7.9. For NEW_UI bundle format only. |
| showBundlesForEditingOrder | List of editable intrinsics and metadata elements in order they should be displayed, separated by commas or returns. Only bundles specified in the "showBundlesForEditing" setting may be referenced here. | Any valid sortable intrinsic or metadata element bundle that is specified in "showBundlesForEditing" | Available as of version 1.7.9. For NEW_UI bundle format only. |
| numPerPage | Controls the number of representations initally loaded. Default is 10. Larger values may degrade performance. | Any integer > 0 | Available as of version 1.7.9 |
| effectiveDateDefault | Default effective date value for newly added representation relationships. Leave blank if you do not wish to set an effective date. | Valid date expression. Use "now" to stam with the current date/time. | |
| dontShowPreferredLabel | Disables display of the representation preferred label in the CLASSIC bundle format when set to a non-zero value. | 0,1 | For CLASSIC bundle format only. |
| dontShowIdno | Disables display of the representation identifier in the CLASSIC bundle format when set to a non-zero value. | 0,1 | For CLASSIC bundle format only. |
| dontShowStatus | Disables display of the representation status value in the CLASSIC bundle format when set to a non-zero value. | 0,1 | For CLASSIC bundle format only. |
| dontShowAccess | Disables display of the representation ac- | 0,1 | For CLASSIC bun- |

### 1.43.2 Metadata element settings

### 1.43.3 Display settings

## 1.44 Expressions

Expressions are statements evaluated by CollectiveAccess to a text, numeric or boolean (*true/false*) value. Expressions can be used to conditionally trigger (or not) elements of an import mapping or display template, where the boolean value returned determines what happens. Values may be generated through use of functions (described in more detail below), comparisons and mathematical operations.

At its simplest an expression is a number or text quantity. These are examples of perfectly valid expressions:

- 5
- "Software is great"

You'll notice in the examples above that numbers are just numbers while text must be enclosed in quotes (single or double). Any quantity that is non-empty and non-zero will evaluate to "true" meaning that 5 = *true* while 0 = *false*. -1 is also *true*, as it is a non-zero value. Any strings besides "" (no text at all) is *true*, even " " (a single space).

While single values are valid expressions, they're usually only useful when used in conjunction with *operators*. Operators are symbols that take two operands (values), perform some operation, and return a new value based upon the operands. There are several types of operators available in expressions:

### 1.44.1 Comparison operators

Comparison operators compare two operands and return *true* or *false*. The most common operator is "=", which returns *true* if the operands are exactly the same, false if they are not. For example, "wood" = "wood" is *true* whereas "wood" = "cement" is not. In an import mapping it is possible to use the "=" operator to check if an input field is a certain value.

Other comparison operators are:

- > greater than
- < less than
- >= greater than or equal
- <= less than or equal
- <> not equal
- != not equal (alternate form)

Greater than/less than operators only work with numeric values. Equal and not equal work with numbers or text.

#### Boolean comparison

As of version 1.7.9 values representing boolean *true* and *false* are available for use in comparisons. These allow you to more easily test the return value of an expression of function using the bare, unquoted word "true" or "false". For example, this expression:

```
dateIsRange("1950's") = false
```

Would return *true* when dateIsRange() returns false, which is useful for importer actions and display templates where specific behaviors are triggered by true expressions.

### 1.44.2 Math operators

With expressions you can perform mathematical operations on numbers using +, -, * and /. These are addition, subtraction, multiplication and division respectively. The + operator also works on text, and will merge two text values together into a single run-on text value. For example:

```
4 + 5
```

will return the value *9*

```
"Julia" + " plus " + "Allison"
```

will return the value *"Julia plus Allison"*

### 1.44.3 Logical operators

It is also possible to string together many expressions into a larger composite expression using the boolean logic operators "AND" and "OR". "AND" returns *true* if, and only if, both operands evaluates as *true*. "OR" returns *true* if, and only if, at least one operand evaluates as *true*. For example:

```
(5 > 10) AND ("seth" = "seth")
```

is false because 5 is not greater than 10, and both expressions need to be true for the composite AND to be true

```
(5 > 10) OR ("seth" = "seth")
```

is true because "seth" = "seth" is true and only one needs to be true for logical OR to return true

**Note:** Prior to version 1.7.9 logical operators were required to be upper-case only. Both upper and lower-case operators are now allowed.

### 1.44.4 Additional comparison operators

The comparison operators shown above are useful but limited. There are a couple of additional ones that are really where the action is :-) They are:

### 1.44.5 The "IN" operator

"IN" lets you compare a value to a list of values. It returns true if ANY value in the list matches the value you are comparing. For example:

```
"Seth" IN ["Julia", "Allison", "Sophie", "Maria", "Angie", "Seth"]
```

returns *true* while

```
"Joe" IN ["Julia", "Allison", "Sophie", "Maria", "Angie", "Seth"]
```

returns *false*.

There is also a related "NOT IN" operator which will return *true* if the value is not in the list.

### 1.44.6 The =~ (regular expression) operator

You can compare a value against a regular expression using the =~ operator. Regular expressions are a very powerful and very flexible pattern matching syntax. At its most basic a regular expression is a simple bit of text that is matched anywhere in the value being compared. For example:

```
"Software is great" =~ /soft/
```

returns *true*.

Note that the regular expression is on the right side of the operator and is enclosed in "/" characters. This is a traditional notation for regular expressions; they are enclosed in the forward slashes to set them off from normal text.

There is also a related !~ operator which will return *true* when the value does not match the regular expression.

### 1.44.7 Variables

This is all well and good, but the above examples are not terribly useful with hardcoded values in them. Where things start getting truly useful is variables. Any source in an import record can be used as a variable by prefixing its name with a "^" character. So if you were importing an Excel spreadsheet and wanted to apply rules when the word "allison" appears anywhere in the value of column 4 you'd write

```
^4 =~ /allison/
```

Similarly, if you want to make sure that the value in the 10th column is equal to "metal" then you use the expression:

```
^10 = "metal"
```

If you wanted to make sure that both conditions applied to a record then you'd use:

```
(^4 =~ /allison/) AND (^10 = "metal")
```

If either would suffice you could use "OR" rather than "AND"

For XML input data the variable names are the XML paths – the exact same thing used in the source specification but with a "^" tacked onto the front.

### 1.44.8 Functions

Functions are black-boxes that you put a number of values into in order to get a single value out of. The expression system current allows the following functions:

| Function | Description | Parameters | Return value | Example |
|---|---|---|---|---|
| abs | Returns the absolute value of a number (eg. changes negative numbers to positive ones); takes a single value as input | | | |
| ceil | Rounds a fractional number up to the next highest integer; takes a single value as input | | | |
| floor | rounds a fractional number down to the next lower integer; takes a single value as input | | | |
| int | Forces a number to be an integer. If the number has a decimal component it is discarded; takes a single value as input | | | |

Continued on next page

Table 6 – continued from previous page

| Function | Description | Parameters | Return value | Example |
|---|---|---|---|---|
| max | Returns the largest value of those passed to it; takes any number of values as input | | | |
| min | Returns the smallest value of those passed to it; takes any number of values as input | | | |
| round | Rounds the number to the closest integer; takes a single value as input | | | |
| random | Returns a random number between zero and the number provided as input ; takes a single value as input | | | |
| rand | Synonym for random | | | |
| current | Evaluates to true if the supplied date expression encompasses the current server date/time [available from version 1.5] | String <date expression> | | |
| future | Evaluates to true if the supplied date expression ‘’ends” any time after the current server date/time. The start date is not considered, so the range may start before or after the current date/time and still evaluate to true [available from version 1.5] | | | |
| wc | Returns number of words (wc = “word count”) in a supplied text value [available from version 1.5] | String <text> | | |
| length | Returns number of characters in a supplied text value [available from version 1.5] | String <text> | | |
| sizeof | Returns number of parameters. useful for counting values. See example below [available from version 1.6] | | | |
| count | Synonym for sizeof | | | |
| age | Calculates age in years. accepts an arbitrary number of parameters greater than 1. It’ll take the earliest and latest dates in the parameter list as start and end of the time span, so you don’t have to worry about the order. If the result is a span of 0 years (e.g. because only 1 date was passed), it’ll retry with the current date added to the list. This is useful to calculate something’s/someone’s current age. [available from version 1.6] | Any number of date expressions | | |
| ageyears | Alias for age [available from version 1.6] | Any number of date expressions | | |
| agedays | Same as age/ageyears, only for days. [available from version 1.6] | Any number of date expressions | | |
| avgdays | Calculates the average length of the time spans passed as parameters. Accepts an arbitrary number of parameters (>1). [available from version 1.6] | Any number of date expressions | | |

Continued on next page

Table 6 – continued from previous page

| Function | Description | Parameters | Return value | Example |
|---|---|---|---|---|
| formatdate | Formats a valid date expression using PHP's [http://php.net/manual/en/function.date.php date() function]. Formats dates as ISO by default but accepts an optional second parameter to specify the format that gets passed to date(). See the PHP documentation for available options. [available from version 1.6] | String <date expression> [String <date format>] | | |
| formatgmdate | Formats a valid date expression in UTC using PHP's [http://php.net/manual/en/function.gmdate.php gmdate() function]. Formats dates as ISO by default but accepts an optional second parameter to specify the format that gets passed to gmdate(). See the PHP documentation for available options. [available from version 1.6] | String <date expression> [String <date format>] | | |
| isvaliddate | Returns true if parameter parses as a valid date [available from version 1.7] | String <date expression> | | |
| date | Parses a natural language date into a pair of historic timestamp values, suitable for mathematical comparison. | String <date expression> | | |
| join | Returns a list of values delimited by the first argument. All other arguments are values. Alias ''implode''. [available from version 1.7] | Any number of string values | | |
| implode | Synonym for join | | | |
| trim | Trims leading and trailing whitespace from a string. [available from version 1.7] | <string> text | | |
| avg | Return average of parameter values. | Any number of numeric values | | |
| sum | Return sum of parameter values | Any number of numeric values | | |
| replace | Replace values using regular expression | String <Perl compatible regular expression> String <replacement value> String <subject value> | | |
| idnoUseCount | Return number of items a value is used as an identifier (idno) for a given table. | String <idno value> String <table> (optional, if omitted defaults to "ca_objects") | | |
| dateIsRange | Return true if date is a range rather than a single day, month or year. [available from version 1.7.9] | String <date expression> | boolean | dateIsRange(1950's) |

To include the function-produced value in your expression just add the function name with a paren-enclosed list of values following. For example:

```
random(10) > 5
```

returns *true* if the random number between 0 and 10 is greater than 5.

- ceil(5.2) returns 6
- floor(5.6) returns 5

- round(5.2) returns 5
- round(5.6) returns 6
- length("hello") returns 5
- sizeof(1,2,3,4) returns 4
- age("23 June 1912", "7 June 1954") returns 41
- age("7 June 1954", "23 June 1912") returns 41 (order doesn't matter)
- age("7 June 1954", "9 May 1945", "23 June 1912") returns 41 ('extra' dates don't matter)
- age("28 January 1985") returns something > 29; 30 if you run it before 28 January 2016
- agedays("23 June 1912", "7 June 1954") returns 15324
- agedays("1912/06/23") returns something > 37653
- avgdays("1912/06/23 - 1954/06/07", "1985/01/28 - 2015/07/24") returns 13229
- avgdays("1945/01/02 - 1945/01/03", "1985/01/28 - 1985/01/29") returns 1
- formatdate("1985/01/28") returns 2015-08-05T14* 28* 31-04* 00. Note that this result can vary based on your time zone setting in setup.php!
- formatgmdate("1985/01/28") returns 1985-01-28T05* 00* 00+00* 00. Note that this result can vary based on your time zone setting in setup.php!
- formatgmdate("1985/01/28", "Y") returns 1985
- trim(" this text has spaces at the end ") returns "this text has spaces at the end"
- join(", ", "Smith", "Bob") returns "Smith, Bob"

### 1.44.9 Parentheses

You may have noticed that parens have been sprinkled through some of the examples. You can use matched parens to group elements of an expression. This makes it easier to read and also ensures that operators are applied in the desired sequence in complex expressions. The three things you need to know about parens are: (1) each paren'ed sub-expression is evaluated as a single unit, before being combined with other sub-expressions (2) you must always match each opening paren with a closing paren and (3) parens don't hurt anything, but can improve readability of the expression so you are encouraged to use them liberally.

## 1.45 Glossaries

### 1.45.1 API Endpoints & Methods

### 1.45.2 Bundles

## Intro

Bundles are elements that can be placed on UI screens, included in search forms or displays. They can be attributes of a specific element set or database fields intrinsic to a specific item type. Bundles can be functional elements that allow cataloguers to establish relationships between items, add and remove items from sets and manage an item's location in a larger hierarchy. Bundles are so named because they are essentially black-boxes that encapsulate various functionality.

Below is a break down of the bundle classes and the properties that are particular to each type.

| Bundle type | Also known as | Description | Example |
|---|---|---|---|
| Basic bundle | Administrative bundle, Intrinsic bundle | Always present regardless of configuration. Single data entry; does not repeat. | access |
| Relationship bundle | Related table | Bundles that create relationships between items. | ca_objects |
| Label bundles | Name or Title bundle | Human-readable short descriptions used for display to identify a record. | preferred_labels |
| Attribute bundles | Metadata element | Any field created by a user. | ca_attribute_elementcode |
| Special bundles | | Bundles that allow a cataloger to manage an item's locations in, for example, sets and hierarchies | hierarchy_location |

## User interface Settings

There are several settings that can be used to configure all bundles, regardless of type, when they are placed on a user interface screen.

| Settings | Description | Default | Values |
|---|---|---|---|
| label | Custom label text to use for this placement of this bundle. | | |
| add_label | Custom text to use for the add button for the placement of this bundle. | | |
| description | Descriptive text to use for help for bundle. Will override descriptive text set for underlying metadata element, if set. Make sure to include a locale specification, i.e. <setting name="description" locale="en_US">XXX</setting> | | |
| readonly | If checked, field will not be editable. | 0 (not read only) | 0 or 1 |
| expand_collapse_value | (Available for v1.5) Controls the expand/collapse behavior when there is at least one value present. While technically available for most bundles, the setting might have no effect for some of the "special" bundles. They have extra settings, see below. | dont_force (default behavior = save expand/collapse state when the user changes it) | dont_force, collapse, expand |
| expand_collapse_no_value | (Available for v1.5) Controls the expand/collapse behavior when there is no value present. While technically available for most bundles, the setting might have no effect for some of the "special" bundles. They have extra settings, see below. | dont_force (default behavior = save expand/collapse state when the user changes it) | dont_force, collapse, expand |

However, there are type-specific settings as well, outlined below.

| | | | | |
|---|---|---|---|---|
| Bundle type | Settings | Description | Default | Values |
| Special (hierarchy_location and hierarchy_navigation only) | open_hierarchy | If checked hierarchy browser will be open when form loads. | 1 (open) | 0 or 1 |
| Special (hierarchy_location and hierarchy_navigation only) | auto_shrink | If enabled hierarchy browser will shrink to height of contents. Version 1.5 and later. | 1 (shrink) | 0 or 1 |
| Special (hierarchy_location, hierarchy_navigation and ca_objects_history only) | expand_collapse | Controls the expand/collapse behavior for this bundle. | dont_force (default behavior = save expand/collapse state when the user changes it) | dont_force, collapse, expand |
| Relationship (ca_list_items only) | restrict_to_lists | Restricts display to items from the specified list(s). Leave all unselected for no restriction. | | list code |
| Relationship (ca_list_items, ca_storage_locations, ca_places only) | useHierarchicalBrowser | If set a hierarchy browser will be used to select the related item rather than an auto-completing text field. | 0 | 0 or 1 |
| Relationship (ca_list_items, ca_storage_locations, ca_places only) | hierarchicalBrowserHeight | The height of the hierarchical browser displayed when theuseHierarchicalBrowser option is set. | 200px | A pixel dimension ending with ‘px’ (eg. 500px) |
| Relationship (ca_objects) | restrictToTermsRelatedToCollection | Will restrict checklist to those terms applied to related collections. | 0 | 0 or 1 |
| Relationship (ca_objects) | restrictToTermsOnCollectionWithRelationshipType | Will restrict checklist to terms related to collections with the specified relationship type. Leave all unselected for no restriction. | | type code |

Here's an example of how some of the settings above would look at the code-level in an xml profile:

```
<placement code="ca_film">
  <bundle>ca_objects</bundle>
       <settings>
         <setting name="restrict_to_types">film</setting>
         <setting name="label" locale="en_US">Related films</setting>
         <setting name="add_label" locale="en_US">Add film</setting>
       </settings>
</placement>
```

## Display Settings

From /app/models/ca_bundle_displays.php

Global display settings:

| Settings | Description | Default | Values |
|---|---|---|---|
| show_empty_values | If checked all values will be displayed, whether there is content for them or not. | 1 | 0 or 1 |

Bundle display settings for all types:

| Settings | Description | Default | Values |
|---|---|---|---|
| label | Custom label text to use for this placement of this bundle. | | Text |

Type-specific bundle display settings:

| Bundle type | Settings | Description | Default | Values |
|---|---|---|---|---|
| Label, Attribute, Relationship | delimiter | Text to place in-between repeating values. | | |
| Label, Attribute | format | Template used to format output. | | |
| Label, Attribute | maximum_length | Maximum length, in characters, of displayed information. | 100 | Characters |
| Relationship | makeEditorLink | If set name of related item will be displayed as a link to edit the item. | 0 (not a link) | 0 or 1 |
| Relationship | restrict_to_relationship_types | Restricts display to items related using the specified relationship type(s). Leave all unselected for no restriction. | | type code |
| Relationship | restrict_to_types | Restricts display to items of the specified type(s). Leave all unselected for no restriction. | | type code |
| Relationship | show_hierarchy | If checked the full hierarchical path will be shown. | 1 (full hierarchy shown) | 0 or 1 |
| Relationship | remove_first_items | If set to a non-zero value, the specified number of items at the top of the hierarchy will be omitted. For example, if set to 2, the root and first child of the hierarchy will be omitted. | 0 | Integers zero or greater based on hierarchy |
| Relationship | hierarchy_order | Determines order in which hierarchy is displayed. | | ASC (top first) DESC (bottom first) |
| Attribute | show_empty_values | If checked all values will be displayed, whether there is content for them or not. | 1 | 0 or 1 |
| Attribute | filter | Expression to filter values with. Leave blank if you do not wish to filter values. | | ^ca_objects.dimensions.Type IN ("with_frame") |

### Search Form Settings

Regardless of type, bundles can take the follow setting when they are used in search forms.

| Settings | Description | Default | Values |
|---|---|---|---|
| label | Custom label text to use for this placement of this bundle. | | Text |
| width | Width, in pixels, of search form elements. | 100px | A pixel dimension ending with 'px' (eg. 500px) |

## 1.45.3 Supported Media File Formats

## Supported Image Formats

| Format | Media processing backend | Comments |
|---|---|---|
| JPEG | GD, ImageMagick, IMagick, GraphicsMagick, Gmagick | |
| GIF | GD, ImageMagick, IMagick, GraphicsMagick, Gmagick | |
| TIFF | ImageMagick, IMagick, GraphicsMagick, Gmagick | |
| PNG | GD, ImageMagick, IMagick, GraphicsMagick, Gmagick | |
| TilePic | GD, ImageMagick, IMagick, GraphicsMagick, Gmagick | |
| Camera RAW | ImageMagick, IMagick, GraphicsMagick, Gmagick | Supports whatever manufacturer formats your installed version of ImageMagick can handle |
| Photoshop PSD | ImageMagick, IMagick, GraphicsMagick, Gmagick | Derivatives for files with layer effects may not be rendered properly |
| JPEG-2000 | ImageMagick, IMagick, GraphicsMagick, Gmagick | |
| DICOM | ImageMagick, IMagick, GraphicsMagick, Gmagick | |
| DPX | ImageMagick, IMagick, GraphicsMagick, Gmagick | Pixel values in non-Tilepic derivatives are adjusted for optimal viewing on computer monitors. |
| OpenEXR | ImageMagick, IMagick, GraphicsMagick, Gmagick | More information on this format can be found here. |
| QTVR (QuickTime VR) | QuicktimeVR | JPEG-format thumbnails are extracted; straight-QTVR is kept as "original" version. The QuicktimeVR plugin requires ffmpeg to be installed. |
| Adobe DNG | ImageMagick, IMagick, GraphicsMagick, Gmagick | |

## Supported Audio Formats

| Format | Media processing backend | Comments |
|---|---|---|
| MP3 | ffmpeg | ffmpeg must be compiled with libLAME |
| AIFF | ffmpeg | |
| WAV | ffmpeg | Including Broadcast-WAVE |
| AAC | ffmpeg | Requires ffmpeg to be compiled with libfaac |
| Ogg Vorbis | ffmpeg | |

### Supported Video Formats

| Format | Media processing backend | Comments |
|---|---|---|
| MPEG-2 | ffmpeg | |
| MPEG-4 | ffmpeg | |
| QuickTime | ffmpeg | |
| WindowsMedia | ffmpeg | |
| FLV | ffmpeg | |
| Ogg Theora | ffmpeg | |
| AVI | ffmpeg | ffmpeg |
| xvid | ffmpeg | Planned |

### Supported Document Formats

| Format | Media processing backend | Comments |
|---|---|---|
| PDF | pdflib (optional); xPDF (optional) | PDFLib-Lite and the PDFLib PECL module will improve processing performance; CA will automatically use PDFLibLite if available. Otherwise the built-in (no installation necessary) Zend_PDF library will be used. CA can only generate thumbnail page images for display if Ghostscript is installed on your system. Can extract text from file for searchability if xPDF is installed. |
| Microsoft Word | AbiWord (optional); Libre Office (optional) | Both .doc (< 2007 file format) and .docx (XML-based format) are supported. Can extract text from file for searchability if AbiWordis installed. Can generate page previews if LibreOffice is installed. |
| Microsoft Powerpoint | Libre Office (optional) | As of version 1.2 XML-based format is supported. Can generate page previews if LibreOffice is installed. |
| Microsoft Excel | Libre Office (optional) | As of version 1.2 XML-based format is supported. Can generate page previews if LibreOffice is installed. |

### Supported Multimedia Formats

| Format | Media processing backend | Comments |
|---|---|---|
| STL | mesh | Standard Tesselation Language |
| PLY | mesh | Polygon File Format |