

## BUILDING SYSTEM INSTALLATION PROFILES (for dummies)

- 1.0 Introduction
  - 2.0 CollectiveAccess Basics
  - 3.0 About Profiles
    - 3.1 Types of Profiles in CollectiveAccess
    - 3.2 Parts of a Profile
      - 3.2.1 Profile Declaration
      - 3.2.2 Locale Definitions
      - 3.2.3 List Definitions
        - 3.2.3.1 System (Structure) Lists
        - 3.2.3.2 Control (User) Lists
      - 3.2.4 Metadata Element Set (Attribute) Definitions
        - 3.2.4.1 Data Types (Attribute Types)
        - 3.2.4.2 Settings
        - 3.2.4.3 Type Restrictions
      - 3.2.5 User Interface Definitions
      - 3.2.6 Relationship Types
  - 4.0 Putting It All Together
    - 4.1 Modifying an Existing Profile
    - 4.2 Building a Profile From Scratch
      - 4.2.1 Begin With the Basics
      - 4.2.2 Study Other Profiles and the Wiki
      - 4.2.3 Work in Sections
    - 4.3 Saving a Profile
  - 5.0 Installation and Testing
  - 6.0 Conclusion
  - 7.0 Resources
- 

### 1. INTRODUCTION

CollectiveAccess is often billed as *collections management software* for museums and archives. However, a more general (and perhaps more accurate) description would be *a highly configurable cataloguing framework*. It can manage information about collections of people or places or events as effectively as it can manage information about collections of objects. As such it is well suited for a wide variety of cataloguing projects beyond collections management.

Unlike most other cataloguing systems, CollectiveAccess doesn't force you to

use a particular set of fields, or even a particular set of cataloguing data entry screens. Virtually everything is configurable. Indeed, just about everything *must* be configured if your CollectiveAccess installation is to be usable.

The freedom CollectiveAccess provides enables you to closely tailor the system to the specific needs of your project. But this freedom comes at a price: for all but the simplest systems it can be a daunting (and exhausting) task to define every single field, screen and relationship.

That said, most users will never have to deal with creating an entire CollectiveAccess configuration from scratch. Numerous configurations developed by the CollectiveAccess user community have been captured for reuse as *installation profiles* – pre-existing installations that you can use as templates or starting points for your own projects. Some of these installation profiles are straightforward implementations of popular metadata standards – DublinCore, PBCore, SPECTRUM, etc. – while others are user-created custom setups for specific use cases – photo archive, catalogue raisonné, film archive, historical society, online exhibition, and more.

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

- **Metadata elements** (aka “fields”). 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 CollectiveAccess’ web-based UI editor, but it is usually convenient (and highly recommended) to define at least a basic set of interfaces in a profile.
- **Lists and vocabularies**. Lists are used extensively in

CollectiveAccess as controlled vocabularies for cataloguing, as drop-down menus 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 your 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

It should be noted that CollectiveAccess is actually a family of applications that work together to provide a seamless cataloguing, collections management and collections publishing platform. Each component serves a specific function and may be downloaded separately. *Providence* is the core cataloguing tool and database application used to manage and describe your collection. The installation profiles you will learn to develop are used by *Providence* to define the cataloguing environment.

Two other components are currently under development (as of October 2009):

- *Pawtucket* is a public-access module providing a friendly web-based search and browse interface to collections data, as well as interactive functionality such as user tagging, commenting, rating and user-generated slideshows. You can use *Pawtucket* to easily make any subset of your collections data publicly available on the web.

- *Woonsocket* is a native iPhone application that enables mobile users to search and browse collections data from any participating CollectiveAccess-based data provider. Since *Woonsocket* is open-source you can use the code as the basis of your own custom mobile search application.

If you hit a roadblock or have a question, there's always help available online, either on the CollectiveAccess support forum (<http://www.collectiveaccess.org/forum>) or by email ([support@collectiveaccess.org](mailto:support@collectiveaccess.org)).

So, let's get started!

## 2.0 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:

<b>Objects</b>	Assets within a collection. Typically these are the physical or born-digital items you are managing.
<b>Entities</b>	People or organizations. These are the creators, artists, donors, publishers and others involved in some way with your collection. Once an entity is created in CollectiveAccess, it can be used in cataloguing for any other type of item (objects, places, collections, etc.)
<b>Places</b>	Physical locations, geographic or otherwise. As with entities, once a place is created it can be used in cataloguing throughout the system. Unlike entities, places are inherently hierarchical allowing you to nest place records within one another emulating the natural structure of place names.
<b>Occurrences</b>	Events or “things.” An occurrence is a catch-all term used by CollectiveAccess to refer to contextual items that may require complex cataloguing but are not entities, places, collections or storage locations. You can use occurrences to support structured authorities for virtually any kind of item. Typical uses for occurrences are to support recording of information about historical events, exhibitions, film productions and bibliographies.

<b>Collections</b>	A defined grouping of objects. The notion of “collection” in CollectiveAccess is a fluid one. Collections can represent physical collections, “virtual” collections of items associated by some criteria, or any other arbitrary grouping. Because collections are first-class items that can take arbitrarily complex cataloguing they can be used for everything from simple grouping of objects to containers for collection-level finding aids.
<b>Lots</b>	A single entry for a group of objects, usually recording basic intake information (donor, date of receipt, etc.) for a donation or acquisition, prior to item-level cataloguing.
<b>Sets</b>	An ordered grouping of objects defined by users for a specific purpose. Unlike collections, sets are ad-hoc groups of items created by a user, or groups of users, for a specific operational purpose (e.g., a working set of candidate objects for an exhibition or a set of entities for which contact information is possibly incorrect). They are not meant for collection-level cataloguing.
<b>Set items</b>	The membership of an item (object, entities, place, collection, etc.) in a given set. Items in sets can take arbitrarily complex cataloguing in a <i>per-set</i> basis if desired. This may seem strange (or like overkill) but it enables the construction of sets where each item can have set-specific captions and links. This makes sets and set items a great tool for construction of slideshows and tours based upon collection content.
<b>Representations</b>	Media (images, video, audio, PDFs) documenting objects. While representations often consist of just the media file itself, they can take arbitrarily complex cataloguing. This allows the addition of captions, access information, rights and reproduction restrictions or just about any other type of information on a representation-specific basis, if needed.
<b>Storage locations</b>	Physical locations where collection objects are stored. Like places, storage locations are hierarchical – you can nest locations within locations to allow notation at any level of specificity (building, room, cabinet, drawer). Each storage location can take arbitrarily complex cataloguing, including access restrictions, map coordinates and other information. Storage locations can be linked to both objects and lots, either directly, or through object event or lot events.
<b>Lists</b>	A flat, single-level or hierarchical group of <i>list items</i> . Lists are used throughout CollectiveAccess in the

	following contexts: (1) as drop-down lists constraining the values of an attribute; (2) as controlled vocabularies whose list items can be associated with objects, entities, places, etc.; and (3) as system lists whose list item values customize CollectiveAccess for specific uses. All lists must be assigned a unique list code, which is used internally in the profile to refer to the list. You can make up list codes for your own lists. System lists have standard codes defined by CollectiveAccess.
<b>List items</b>	The individual entries that comprise a list. All list items have an intrinsic value and an identifier, as well as one or more text labels used for display.
<b>Object events</b>	An event in the life-cycle of an object. Each event includes a planned date as well as an actual event date recording when the event occurred. Different types of events (e.g., conservation events, movement events, loan events) can be recorded by defining object event types and configuring the appropriate metadata elements for each type.
<b>Lot events</b>	An event in the life-cycle of a lot. Each event includes a planned date as well as an actual event date recording when the event occurred. Different types of events (e.g., acquisition events, deaccession events) can be recorded by defining lot event types and configuring the appropriate metadata elements for each type.

Installation profiles are written in the same notation as CollectiveAccess configuration files (see <http://wiki.collectiveaccess.org/index.php?title=ConfigurationFileSyntax> for a full description of the syntax). A profile file is composed of key-value pairs. Keys are simple alphanumeric text expressions while 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 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 arrays 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.

As you write your profile it may help to keep in mind the special meanings of the following characters:

<b>=</b>	Used to separate keys from values. In a configuration profile the key is typically either a unique code for whatever you're configuring or the name of a value you're defining.
<b>{ }</b>	Used to open and close blocks of configuration.
<b>,</b>	Separates elements within a profile, list or associative array.
<b>1</b>	When used as a value, indicates "True" or "Yes" – will allow whatever action.
<b>0</b>	When used as a value, indicates "False" or "No" – will deny whatever action.

### 3.0 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:

- a. 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.
- b. 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.
- c. 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.

### **3.1 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)
- Surfing Heritage Foundation, San Clemente, CA, USA, (Museum 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
- SPECTRUM
- MARC (in development)
- DarwinCore (in development)
- VRACore (in development)

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.

## 3.2 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. 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 sections.

### 3.2.1 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_name = ![Standard] DublinCore
profile_description = Use this profile if you want a system that is
                    compliant with baseline DublinCore
profile_use_for_configuration = 1
profile_base = base
profile_info_url = http://www.example.com
```

The text to the left of the '=' on each line is a "key" and the text on the right side is a "value." The profile declaration consists solely of simple value assignments. The first line, for example, assigns "[Standard] DublinCore" to the key "profile\_name." CollectiveAccess uses the values assigned to different keys in various ways. The value of "profile\_name" is used to distinguish the profile in a list of profiles. The value of "profile\_description" is used to describe the profile when a longer description is required.

Two keys in the declaration deserve further explanation. The "profile\_base" key 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 profile\_base key to the file name of the profile *without* the .profile extension.

The “profile\_use\_for\_configuration” key determines whether the profile can be used to install a working system. This is most often used to prevent a base profile from being displayed as a profile option in the CollectiveAccess installer.

### 3.2.2 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\\_Translation](http://wiki.collectiveaccess.org/index.php?title=Creating_a_Translation) for a current list of application translations.

*Coded for English:*

```
locales = {  
    en_US = English (US)  
}
```

*Coded for German and English:*

```
locales = {  
    de_DE = Deutsch (DE),  
    en_US = English (US)  
}
```

Let’s take a closer look at the above code. The first line states the part of the profile, followed by an equal sign and curly bracket. This introduces a very key concept for nearly all the coding within profiles: the LEFT and RIGHT of the equal sign relationship. This opening line of code is basically saying, “We’re defining the locales (languages) now and they are...” The lines after the opening curly bracket “{” define which languages are going to be used in the system: “...and they are English and German.” English and German are defined by the code (“de\_DE” and “en\_US”) on the LEFT, and the name on the right. This structure is repeated throughout the profile.

### 3.2.3 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 drop-down options to control content and lists that defined controlled vocabularies that can

be used for descriptive cataloguing.

### 3.2.3.1 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.

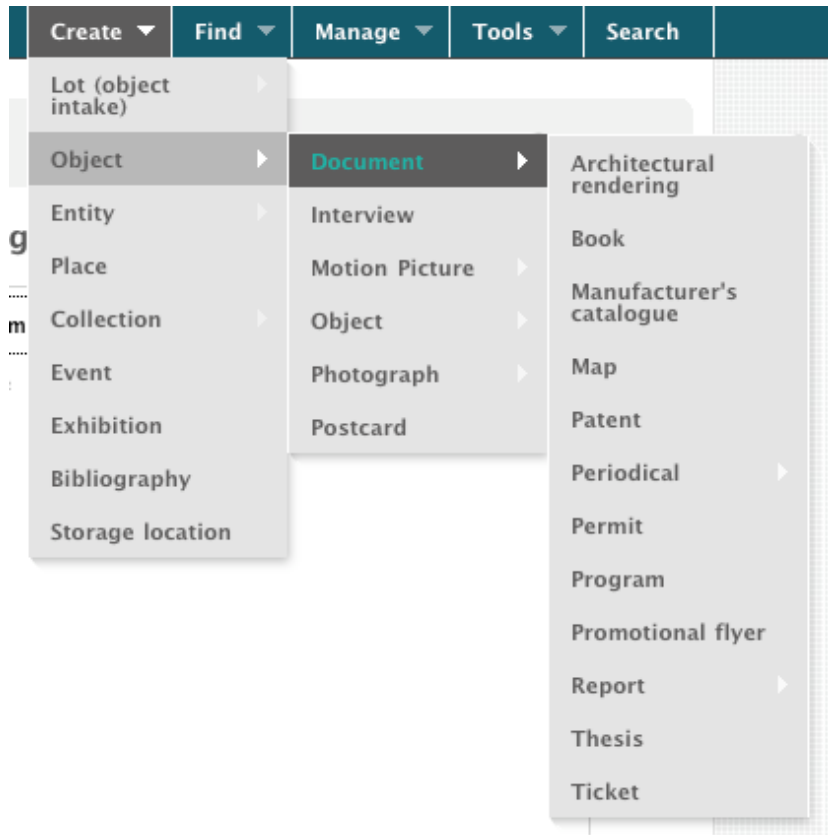


Figure 1

For CollectiveAccess to function properly, 33 types of system lists need to be present and defined for *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. These lists are:

List code	Description
object_sources	Populates object “source” drop-down list. Each object record may optionally have one (and only one) designated source. This is typically used to indicate

	from where a record was imported/obtained, but can be used for any non-repeating list-constrained object value.
object_types	Defines the set of object types supported by the system. Each type can have it's a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together.
object_statuses	Populates object "accession status" drop-down list. Each object record may optionally have one (and only one) accession status. This is typically used to indicate whether an object is accessioned, non-accessioned, a loan, etc.
object_label_types	Populates the label types drop-down list for object labels (object titles are referred to as "labels" internally). Label types are used to optionally distinguish different types of non-preferred labels.
object_acq_types	Populates the object acquisition type drop-down list. Each object record may have one (and only one) acquisition type. This is typically used to indicate how an object was acquired for the collection.
object_lot_types	Defines the set of lot types supported by the system (e.g., gifts, bequests, purchases). Each type can have it's a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together.
object_lot_statuses	Populates lot "lot status" drop-down list. Each lot record may optionally have one (and only one) lot status. This is typically used to indicate whether an object is accessioned, non-accessioned, pending accession, etc.
object_lot_label_types	Populates the label types drop-down list for lot labels (lot titles are referred to as "labels" internally). Label types are used

	to optionally distinguish different types of non-preferred labels.
entity_types	Defines the set of entity types supported by the system (e.g., individuals, organizations, families). Each type can have it's a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together.
entity_sources	Populates entity "source" drop-down list. Each entity record may optionally have one (and only one) designated source. This is typically used to indicate from where a record was imported/obtained, but can be used for any non-repeating list-constrained object value.
entity_label_types	Populates the label types drop-down list for entity labels (entity names are referred to as "labels" internally). Label types are used to optionally distinguish different types of non-preferred labels.
place_types	Defines the set of place types supported by the system (e.g., continents, countries, states, provinces, counties). Each type can have a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together.
place_hierarchies	Defines available place hierarchies in the place editor. If you wish to use the place names authority, you must define at least one entry in this list.
place_sources	Populates place "source" drop-down list. Each place record may optionally have one (and only one) designated source. This is typically used to indicate from where a record was imported/obtained, but can be used for any non-repeating list-constrained object value.
place_label_types	Populates the label types drop-down list for place labels (place names are referred to as "labels" internally). Label types are used to optionally distinguish different types of non-preferred labels.

occurrence_types	Defines the set of occurrence types supported by the system (e.g., exhibitions, historic events, bibliography). Each type can have a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together. Unlike other authorities, where types are presented as sub-divisions, each occurrence type is presented as an independent authority. This allows you to create custom authority lists (and editors) by simply creating a new occurrence type.
occurrence_sources	Populates occurrence “source” drop-down list. Each occurrence record may optionally have one (and only one) designated source. This is typically used to indicate from where a record was imported/obtained, but can be used for any non-repeating list-constrained object value.
occurrence_label_types	Populates the label types drop-down list for occurrence labels (occurrence names are referred to as “labels” internally). Label types are used to optionally distinguish different types of non-preferred labels.
collection_types	Defines the set of collection types supported by the system (e.g., external collection, virtual collection, in-house collection). Each type can have a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together.
collection_sources	Populates collection “source” drop-down list. Each collection record may optionally have one (and only one) designated source. This is typically used to indicate from where a record was imported/obtained, but can be used for any non-repeating list-constrained object value.

collection_label_types	Populates the label types drop-down list for collection labels (collection names are referred to as “labels” internally). Label types are used to optionally distinguish different types of non-preferred labels.
storage_location_types	Defines the set of storage location types supported by the system (e.g., buildings, floors, rooms, cabinets, drawers). Each type can have a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together.
storage_location_label_types	Populates the label types drop-down list for storage location labels (storage location names are referred to as “labels” internally). Label types are used to optionally distinguish different types of non-preferred labels.
object_representation_label_types	Populates the label types drop-down list for object representation labels (object representation titles are referred to as “labels” internally). Label types are used to optionally distinguish different types of non-preferred labels.
representation_annotation_label_types	Populates the label types drop-down list for representation annotation labels (annotation titles are referred to as “labels” internally). Label types are used to optionally distinguish different types of non-preferred labels.
list_item_types	Defines the set of list item types supported by the system (e.g., terms, headings, facets). Each type can have a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together. Types are optional for list items. You only need to define them if you need to distinguish between various types of items in your lists.
list_item_label_types	Populates the label types drop-down list for object labels. Label types are used to optionally distinguish different types

	of non-preferred labels.
object_event_types	Defines the set of object event types supported by the system (e.g., loan, conservation, loss). Each type can have a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together.
object_event_label_types	Populates the label types drop-down list for object event labels (object event names are referred to as “labels” internally). Label types are used to optionally distinguish different types of non-preferred labels.
object_lot_event_types	Defines the set of lot event types supported by the system (e.g., appraisal, inspection, fumigation). Each type can have a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together.
set_types	Defines the range of set types supported by the system (e.g., curated sets, user generated sets, online exhibitions). Each type can have a distinct set of metadata attributes bound to it. You can structure the list hierarchically to group related types together.
access_statuses	Populates the “access” drop-down menu present for most item types (objects, entities, places, occurrences, collections, list items, object representations, representation annotations). The “access” value is typically used to determine whether a specific item should be displayed to a user of a public-access front-end. For example, the “Pawtucket” front-end will only display items with an access value of “1”
workflow_statuses	Populates the “status” drop-down menu present for all item types (objects, entities, places, occurrences,

	collections, list items, object representations, representation annotations, storage locations). The “status” value is typically used to indicate at what stage in the workflow a given item is (e.g., “In editing”, “Needs approval”, “Approved”)
--	--

### 3.2.3.2 Control (User) Lists

Control lists allow you to create drop-down menus in a cataloging record. Any kind of control list can be designed, coded and used. The first step is to think about what kind of controlled options you want for your project. Maybe a list of media formats or a list of genre headings? If you want to restrict a particular set of data, a drop-down list will provide consistent and quick data entry. Let’s look closely at a control list for formats in Dublin Core:

```

dc_format = {
  preferred_labels = {
    en_US = {
      name = Dublin Core Format
    }
  },
  is_hierarchical = 0,
  use_as_vocabulary = 1,
  items = {
    option0 = {
      is_enabled = 1,
      is_default = 1,
      preferred_labels = {
        en_US = {
          name_singular = -,
          name_plural = -
        }
      }
    },
    application = {
      is_enabled = 1,
      is_default = 0,
      preferred_labels = {
        en_US = {
          name_singular = Application,
          name_plural = Applications
        }
      }
    },
    audio = {
      is_enabled = 1,
      is_default = 0,

```

```

        preferred_labels = {
            en_US = {
                name_singular = Audio,
                name_plural = Audio
            }
        }
    },

```

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:

- Opening header element “dc\_format” – the code name for this list.
- Preferred 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).
- *is\_hierarchical* & *use\_as\_vocabulary* – control how data will display and be used in the editing user interface. If *is\_hierarchical* is set to 1, then the list can be a multi-level hierarchy. If *use\_as\_vocabulary* is set to 1, then the list is considered to be a controlled vocabulary and will be included in vocabulary term searches.
- *items* = { - opens the list of items
- *option0*, *application*, & *audio* – are items in the list. Note that the key for each item **must** be unique within the list.
- *is\_enabled*, *is\_default*, & *preferred\_labels* – define how items in the list will display. If *is\_enabled* is set to 0, then the item will display but not be selectable. If *is\_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 *is\_default*. *preferred\_labels* sets 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 create drop-down menus.

### 3.2.4 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 <http://wiki.collectiveaccess.org/index.php?title=AttributeTypes>

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:

```
description = {
    datatype = Text,
    preferred_labels = {
        en_US = {
            name = Description,
            description = An account of the resource.
        }
    },
    settings = {
        fieldWidth = 80,
        fieldHeight = 5,
        minChars = 0,
        maxChars = 65535
    },
    documentation_url = http://dublincore.org/documents/dcmi-
terms/#terms-description,

    type_restrictions = {
        r1 = {
```

```

        table = ca_objects,
        type = ,
        settings = {
            minAttributesPerRow = 0,
            maxAttributesPerRow = 255,
            minimumAttributeBundlesToDisplay = 1
        }
    },
}

```

The element set opens with its “code name” (in this example it is “description”). As with lists and locales, the code name opens this set and will be referred to later in the profile. **Important:** make sure that each element set you defined has a unique code name within the profile. If you create two element sets with the same code name, only the last defined element set will be available.

### 3.2.4.1 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 15 attribute types to choose from:

Container	Unlike all other attribute types, containers do not represent data values. Rather their sole function is to organize attributes into groups for display.
Text	Represents a free-text value and is the most common datatype.
DateRange	Represents an historic date range accepting date/time range expressions in the CollectiveAccess TimeExpressionParser module.
List	Represents a value chosen from a drop-down list populated with values from a specified list as defined in the <i>ca_lists</i> table.
Geocode	Represents one or more latitude/longitude coordinates. Non-coordinate entries are converted to coordinates using the Google Maps Geocoding service, which works well for most full and partial addresses worldwide.
URL	Accepts a properly formatted URL value.
Currency	Accepts a currency value composed of a currency specifier and a decimal number.
Relationship	Provides an attributes-based authority lookup mechanism ( <b>experimental</b> )
Length	Accepts length measurements in metric, English and points units (the latter being typographical points).
Weight	Accepts weight measurements in metric and English units.
Time code	Accepts time offsets in a number of time code formats.
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 part and optional exponential part.
LCSH	Library of Congress Subject Heading value. Takes LCSH heading or first part of heading (the LCSH search service only supports truncation searches), does a lookup and returns a list of possible matches. Selected LCSH heading is stored as both text and the service URL identifier.
GeoNames	GeoNames value. Takes search text, passes it to the GeoNames search service and provides a drop-down list with search results (ordered by score). Selected GeoName is stored as both text (name, country, continent and ID) and the service URL identifier.
File	Uploaded file. CA will try to identify the file and extract limited metadata, but even if it can't identify the file it will accept and store it.
Media	Uploaded media (image, sound video). CA will try to identify the file, extract metadata and create derivatives as configured in <code>media_processing.conf</code> . If CA cannot identify and parse the file it will reject it.

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

```
pbcoreFormatPhysical_video = {
    datatype = List,
    preferred_labels = {
        en_US = {
            name = Format Physical: Video,
            description = "Use the descriptor
formatPhysical to identify the format of a particular version or
rendition of a media item."
        }
    },
    list = pbcore_format_physical_types_video,
```

In this element set the code name is *pbcoreFormatPhysical\_video*, its datatype is *List*, the preferred labels are in English and are *Format Physical: Video*. 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 its *pbcore\_format\_physical\_types\_video*. This links the element set with its corresponding *List* in list definitions creating the necessary code for the drop-down menu style controlled vocabulary.

### 3.2.4.2 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.

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.

Supported settings for current attribute types are:

<b>Datatype</b>	<b>Settings</b>
Container	<i>doesNotTakeLocale</i> (defines whether element take locale specification; values are 0 or 1)
Text	<p><i>minChars</i> (defines the minimum number of characters valid for input into the element; set greater than zero to make element mandatory; valid values are positive integers; default is zero characters)</p> <p><i>maxChars</i> (defines the maximum number of characters valid for input into the element; valid values are positive integers; default is 65535 characters)</p> <p><i>regex</i> (a Perl-compatible regular expression to use for validation of input. Input not matching the expression will be rejected. Do not include the leading and trailing delimiter characters (typically "/") in your expression. Leave blank or do not set if you don't want to validate with a regex.)</p> <p><i>fieldWidth</i> (the width of the text entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the text entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the text element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
DateRange	<p><i>dateRangeBoundaries</i> (The range of dates that are valid for the element. Dates outside the range will be rejected. Leave blank or don't set if you don't require restrictions; valid values are any valid date expression; see <a href="http://wiki.collectiveaccess.org/index.php?title=DateAndTimeFormats">http://wiki.collectiveaccess.org/index.php?title=DateAndTimeFormats</a> for a description of valid input formats.)</p> <p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p>

	<p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the date range element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
List	<p><i>listWidth</i> (the width, in characters, of the list when displayed in a user interface.; valid values are integers greater than zero; default is 40 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the date range element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
Geocode	<p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the date range element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
URL	<p><i>minChars</i> (defines the minimum number of characters valid for input into the element; set greater than zero to make element mandatory; valid values are positive integers; default is zero characters)</p> <p><i>maxChars</i> (defines the maximum number of characters valid for input into the element; valid values are positive integers; default is 65535 characters)</p> <p><i>regex</i> (a Perl-compatible regular expression to use for validation of input. Input not matching the expression will be rejected. Do not include the leading and trailing delimiter characters (typically "/" in your expression. Leave blank or do not set if you don't want to</p>

	<p>validate with a regex.)</p> <p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the text element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
Currency	<p><i>minValue</i> (defines the minimum numeric value valid for input into the element; set greater than zero to make element mandatory; valid values are any numeric value; default is zero)</p> <p><i>maxValue</i> (defines the maximum numeric value valid for input into the element; valid values are any numeric value; default to not restrict by maximum value)</p> <p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the text element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
Relationship	<p><i>RelTable</i> (Relationship Table - for example ca_objects_x_entities)</p> <p><i>RelType</i> (Relationship Type - an existing relationship type defined within the given relationship table; valid values are positive integers representing ca_relationship_types type_id values)</p> <p><i>CreateLink</i> (Whether or not a create item link should be set for this field; valid values are 0 and 1)</p> <p><i>RefOnly</i> (Determines whether or not an autocomplete field is shown;</p>

	<p>valid values are 0 and 1)</p> <p><i>RightItemType</i> (The type name for the right hand item - for example, individual is a typename for ca_entities)</p>
Length	<p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the date range element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
Weight	<p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the date range element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
Time code	<p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the date range element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
Integer	<p><i>minChars</i> (defines the minimum number of characters valid for input into the element; set greater than zero to make element mandatory; valid values are positive integers; default is zero characters)</p>

	<p><i>maxChars</i> (defines the maximum number of characters valid for input into the element; valid values are positive integers; default is 65535 characters)</p> <p><i>regex</i> (a Perl-compatible regular expression to use for validation of input. Input not matching the expression will be rejected. Do not include the leading and trailing delimiter characters (typically "/" in your expression. Leave blank or do not set if you don't want to validate with a regex.)</p> <p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
Numeric	<p><i>minChars</i> (defines the minimum number of characters valid for input into the element; set greater than zero to make element mandatory; valid values are positive integers; default is zero characters)</p> <p><i>maxChars</i> (defines the maximum number of characters valid for input into the element; valid values are positive integers; default is 65535 characters)</p> <p><i>regex</i> (a Perl-compatible regular expression to use for validation of input. Input not matching the expression will be rejected. Do not include the leading and trailing delimiter characters (typically "/" in your expression. Leave blank or do not set if you don't want to validate with a regex.)</p> <p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the element is the only element in the</p>

	<p>element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
LCSH	<p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the date range element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>
GeoNames	<p><i>fieldWidth</i> (the width of the entry form field in characters; valid values are integers greater than zero; default is 40 characters)</p> <p><i>fieldHeight</i> (the height of the entry form field in rows; valid values are integers greater than zero; default is 1 characters)</p> <p><i>canBeUsedInSort</i> (defines if element set can be used in sort of search results; only valid if the date range element is the only element in the element set; values are 0 or 1)</p> <p><i>doesNotTakeLocale</i> (defines whether element takes locale specification; values are 0 or 1)</p>

### 3.2.4.3 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.

```

type_restrictions = {
    r1 = {
        table = ca_objects,
        type = moving_images,
        settings = {
            minAttributesPerRow = 0,
            maxAttributesPerRow = 255,
            minimumAttributeBundlesToDisplay = 1
        }
    }
}

```

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. The following table names are used for the item types:

Item type	Table name
Objects	ca_objects
Lots	ca_object_lots
Entities	ca_entities
Places	ca_places
Occurrences	ca_occurrences
Collections	ca_collections
Storage locations	ca_storage_locations
Object representations	ca_object_representations
Representation annotations	ca_representation_annotations
Lists	ca_lists
List items	ca_list_items
Sets	ca_sets
Set items	ca_set_items
Object events	ca_object_events
Lot event	ca_object_lot_events

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.
minimumAttributeBundles ToDisplay	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.

### 3.2.5 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, additional data on the second screen, and multimedia on the third. 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.

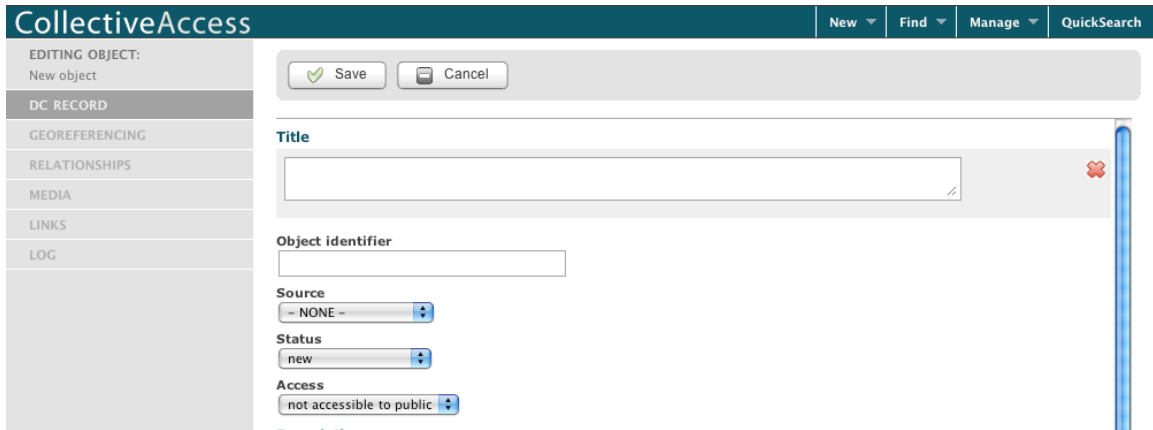


FIGURE 2

To organize the data fields on each screen, bundles are listed out in a *bundle list*:

```

bundles = {
  preferred_labels = {
    bundle = preferred_labels,
    label = {
      en_US = Title
    },
    add_label = {
      en_US = Add title
    }
  },
  idno = { bundle = idno },
  source_id = { bundle = source_id },
  status = { bundle = status },
  access = { bundle = access },
  ca_attribute_description = { bundle = ca_attribute_description},
  ca_attribute_dc_type = { bundle = ca_attribute_dc_type},
  ca_attribute_dc_format = { bundle = ca_attribute_dc_format}
}

```

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. Bundle names for other user interface elements are listed in the following tables.

**For objects (ca\_objects):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting object preferred title
non_preferred_labels	Fields for setting object non-preferred (alternate) titles
idno	Object identifier field (often used for current accession number).
locale_id	Object record locale drop-down
item_status_id	Object status value based upon values in <i>object_statuses</i> list
acquisition_type_id	Object acquisition value based upon values <i>object_acq_types</i> list
source_id	Object source value based upon values <i>object_sources</i> list
extent	Numeric extent
extent_units	Units of extent
access	Control of public access using values defined in the <i>access_statuses</i> list
status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_object_representations	Adds controls for adding, updating removing and viewing linked media
ca_objects	Adds controls for linking to other objects
ca_entities	Adds controls for linking objects to entities
ca_places	Adds controls for linking objects to places
ca_occurrences	Adds controls for linking objects to occurrences
ca_collections	Adds controls for linking objects to collections
ca_storage_locations	Adds controls for linking objects to storage location hierarchy
ca_object_lots	Adds controls for linking object to lot
ca_object_events	Adds controls for creating and managing object event records
ca_list_items	Adds controls for linking terms in

	controlled vocabulary lists to objects
ca_sets	Adds controls for adding and removing object from sets
hierarchy_navigation	Adds user interface display for hierarchy object is part of
hierarchy_location	Adds user interface to change location of object in a hierarchy of objects.

**For lots (ca\_object\_lots):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting lot preferred title
non_preferred_labels	Fields for setting lot non-preferred (alternate) titles
ldno_stub	Lot identifier field (often used for current accession number).
lot_status_id	Lot status value based upon values in <i>object_lot_statuses</i> list
extent	Numeric extent
extent_units	Units of extent
access	Control of public access using values defined in the <i>access_statuses</i> list
status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_objects	Adds controls for linking lots to objects
ca_entities	Adds controls for linking lots to entities
ca_places	Adds controls for linking lots to places
ca_occurrences	Adds controls for linking lots to occurrences
ca_collections	Adds controls for linking lots to collections
ca_storage_locations	Adds controls for linking lots to storage location hierarchy
ca_object_lot_events	Adds controls for creating and managing lot event records
ca_list_items	Adds controls for linking terms in controlled vocabulary lists to lots

**For entities (ca\_entities):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting entity preferred name
non_preferred_labels	Fields for setting entity non-preferred (alternate) names
idno	Entity identifier field
locale_id	Entity record locale drop-down
source_id	Entity source value based upon values <i>entity_sources</i> list
access	Control of public access using values defined in the <i>access_statuses</i> list
status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_objects	Adds controls for linking objects to entities
ca_entities	Adds controls for linking entities to other entities
ca_places	Adds controls for linking entities to places
ca_occurrences	Adds controls for linking entities to occurrences
ca_collections	Adds controls for linking entities to collections
ca_list_items	Adds controls for linking terms in controlled vocabulary lists to entities

**For places (ca\_places):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting place preferred name
non_preferred_labels	Fields for setting place non-preferred (alternate) names
idno	Place identifier field
locale_id	Place record locale drop-down
source_id	Place source value based upon values <i>place_sources</i> list
access	Control of public access using values defined in the <i>access_statuses</i> list

status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_objects	Adds controls for linking objects to places
ca_entities	Adds controls for linking places to entities
ca_places	Adds controls for linking places to other places
ca_occurrences	Adds controls for linking places to occurrences
ca_collections	Adds controls for linking places to collections
ca_list_items	Adds controls for linking terms in controlled vocabulary lists to places
hierarchy_navigation	Adds user interface display for place hierarchy
hierarchy_location	Adds user interface to change location of place in hierarchy

**For occurrences (ca\_occurrences):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting occurrence preferred name
non_preferred_labels	Fields for setting occurrence non-preferred (alternate) names
idno	Occurrence identifier field
locale_id	Occurrence record locale drop-down
source_id	Occurrence source value based upon values <i>occurrence_sources</i> list
access	Control of public access using values defined in the <i>access_statuses</i> list
status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_objects	Adds controls for linking objects to occurrences
ca_entities	Adds controls for linking occurrences to entities

ca_places	Adds controls for linking occurrences to places
ca_occurrences	Adds controls for linking occurrences to other occurrences
ca_collections	Adds controls for linking occurrences to collections
ca_list_items	Adds controls for linking terms in controlled vocabulary lists to occurrences

**For collections (ca\_collections):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting collection preferred name
non_preferred_labels	Fields for setting collection non-preferred (alternate) names
idno	Collection identifier field
locale_id	Collection record locale drop-down
source_id	Collection source value based upon values <i>collection_sources</i> list
access	Control of public access using values defined in the <i>access_statuses</i> list
status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_objects	Adds controls for linking objects to collections
ca_entities	Adds controls for linking collections to entities
ca_places	Adds controls for linking collections to places
ca_occurrences	Adds controls for linking collections to occurrences
ca_collections	Adds controls for linking collections to other collections
ca_list_items	Adds controls for linking terms in controlled vocabulary lists to collections
hierarchy_navigation	Adds user interface display for hierarchy collection is part of
hierarchy_location	Adds user interface to change location of collection in a hierarchy

	of collections
--	----------------

**For storage locations (ca\_storage\_locations):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting storage location preferred name
non_preferred_labels	Fields for setting storage location non-preferred (alternate) names
status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_objects	Adds controls for linking objects to storage locations
hierarchy_navigation	Adds user interface display for storage location hierarchy
hierarchy_location	Adds user interface to change location of storage location in hierarchy

**For object representations (ca\_object\_representations):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting representation preferred title
non_preferred_labels	Fields for setting representation non-preferred (alternate) title
locale_id	Collection record locale drop-down
access	Control of public access using values defined in the <i>access_statuses</i> list
status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_entities	Adds controls for linking representations to entities
ca_places	Adds controls for linking representations to places
ca_occurrences	Adds controls for linking representations to occurrences
ca_representation_annotations	Adds controls for adding annotations to a representation. The type of annotation depends upon the type of media. For example, if

	the media was time-based (audio or video) then the time-based cataloguing UI would be included in the editing by using this bundle.
--	---

**For lists (ca\_lists):**

Bundle name	Description
preferred_labels	Fields for setting list preferred name
non_preferred_labels	Fields for setting list non-preferred (alternate) names
list_code	Field for unique list code
is_system_list	Check box to determine if list is a “system” list or not.
is_hierarchical	Check box to determine if list is hierarchical or not.
use_as_vocabulary	Check box to determine if list can be used as a vocabulary.

**For list items (ca\_list\_items):**

Bundle name	Description
preferred_labels	Fields for setting list item preferred name
non_preferred_labels	Fields for setting list item non-preferred (alternate) names
idno	Place identifier field
item_value	Underlying value of item (independent from the label that is displayed to the user)
is_enabled	Checkbox to determine if list item is selectable or not
is_default	Checkbox to set default item for list
access	Control of public access using values defined in the <i>access_statuses</i> list
status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_objects	Adds controls for linking objects to list items
ca_entities	Adds controls for linking list items to entities
ca_places	Adds controls for linking list items to

	places
ca_occurrences	Adds controls for linking list items to occurrences
ca_collections	Adds controls for linking list items to collections
ca_list_items	Adds controls for linking list items in to other list items
hierarchy_navigation	Adds user interface display for list hierarchy
hierarchy_location	Adds user interface to change location of item in hierarchy

**For sets (ca\_sets):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting set preferred name
table_num	Control for setting what kind of items (objects, entities, places, etc.) the set contains
set_code	Field for unique code identifying set
access	Control of public access using values defined in the <i>access_statuses</i> list
status	Indication of current workflow status as defined in the <i>workflow_statuses</i> list
ca_set_items	Adds controls for adding of items to a set

**For set items (ca\_set\_items):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting set preferred name

**For object events (ca\_object\_events):**

<b>Bundle name</b>	<b>Description</b>
preferred_labels	Fields for setting event preferred title
planned_datetime	Field for date range value indicating when event is planned to occur
event_datetime	Field for date range value indicating when event actually occurred
ca_entities	Adds controls for linking object

	events to entities
ca_places	Adds controls for linking object events to places
ca_occurrences	Adds controls for linking object events to occurrences
ca_storage_locations	Adds controls for linking object events to storage locations

**For lot events (ca\_object\_lot\_events):**

Bundle name	Description
preferred_labels	Fields for setting event preferred title
planned_datetime	Field for date range value indicating when event is planned to occur
event_datetime	Field for date range value indicating when event actually occurred
ca_entities	Adds controls for linking lot events to entities
ca_places	Adds controls for linking lot events to places
ca_occurrences	Adds controls for linking lot events to occurrences
ca_storage_locations	Adds controls for linking lot events to storage locations

Each bundle you add to a user interface can take optional settings. Currently only a few settings are supported, although the list is sure to grow in the near future.

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. The bundle list example above with the a non-default description attribute labels in both English and German would look like this:

```

bundles = {
    preferred_labels = {
        bundle = preferred_labels,
        label = {
            en_US = Title
        },
        add_label = {
            en_US = Add title
        }
    },
    idno =          { bundle = idno },
    source_id =     { bundle = source_id },

```

```

status =          { bundle = status },
access =          { bundle = access },
ca_attribute_description = {
    bundle = ca_attribute_description,
    label = {
        en_US = Narrative description,
        de_DE = Beschreibung
    },
    add_label = {
        en_US = Add another description,
        de_DE = Addieren einen Beschreibung
    }
},
ca_attribute_dc_type =          { bundle = ca_attribute_dc_type},
ca_attribute_dc_format =       { bundle = ca_attribute_dc_format}
}

```

Note that the preferred labels bundle also overrides the default labels, but only includes English-language versions.

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:

```

bundles = {
    ca_entities = {
        bundle = ca_entities,
        restrict_to_type = ind
    }
}

```

### 3.2.6 Relationship Types

CollectiveAccess creates relationships between records that are qualified by descriptive types. These relationship types 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 following list is critical:

ca_collections_x_collections	ca_objects_x_object_events
ca_collections_x_vocabulary_terms	ca_objects_x_object_representations
ca_entities_x_collections	ca_representations_x_places
ca_entities_x_entities	ca_representations_x_vocabulary_terms
ca_entities_x_occurrences	ca_objects_x_objects
ca_entities_x_places	ca_objects_x_occurrences
ca_entities_x_vocabulary_terms	ca_objects_x_places
ca_list_items_x_list_items	ca_objects_x_storage_locations
ca_object_events_x_entities	ca_objects_x_vocabulary_terms
ca_object_events_x_occurrences	ca_occurrences_x_collections
ca_object_events_x_places	ca_occurrences_x_occurrences
ca_object_events_x_storage_locations	ca_occurrences_x_vocabulary_terms
ca_object_events_x_vocabulary_terms	ca_places_x_collections
ca_object_lot_events_x_storage_locations	ca_places_x_occurrences
ca_object_lot_events_x_vocabulary_terms	ca_places_x_places
ca_object_lots_x_collections	ca_places_x_vocabulary_terms
ca_object_lots_x_entities	ca_representation_annotations_x_entities
ca_object_lots_x_occurrences	ca_representation_annotations_x_objects
ca_object_lots_x_places	ca_representation_annotations_x_occurrences
ca_object_lots_x_storage_locations	ca_representation_annotations_x_places
ca_object_lots_x_vocabulary_terms	ca_representation_annotations_x_vocabulary_terms
ca_objects_x_collections	ca_representations_x_entities
ca_objects_x_entities	ca_representations_x_occurrences

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:

```
relationship_types = {
  ca_objects_x_entities = {
    types = {
      creator = {
        is_default = 1,
        preferred_labels = {
          en_US = {
            typename = created by,
            typename_reverse = is creator
          }
        },
        subtype_left = ,
        subtype_right =
      },
    },
  },
}
```

Each of these bundles implements a specific relationship and can repeat as often as is needed. You can define as many Relationship Types for each relationship needed to express the unique linkages between types in your collection.

#### 4.0 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 that can be used for dropdown menus. 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

#### 4.1 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](http://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!

## 4.2 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.

### 4.2.1 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. This is a *Profile Schema Breakdown*, (also known as a metadata application profile). **See the PSBs for existing profiles on our wiki for examples.**

### 4.2.2 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 text files for profiles available in your CollectiveAccess installation can be viewed at:

```
install/profiles/nameofprofile.profile
```

The CollectiveAccess wiki (<http://wiki.collectiveaccess.org>) is also a great resource. There you can read about existing profiles and find more in-depth technical documentation about the topics discussed in this introductory manual.

### 4.2.3 Work in Sections

There are 5 sections that open “{“ and close “}” in every profile and each uses aspects from a section or builds upon the next section. Begin with the first section, and move through one part at a time taking note of code names and formatting as you go along. Working slowly and carefully will pay off later during installation and testing.

## 4.3 Saving a Profile

When saving your new profile, be sure to end the filename with the ".profile"

extension. Save your file in:

```
install/profiles
```

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

## 5.0 Installation and Testing

To install and test your profile, open your browser to your installation webpage. 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. Then click on the "begin" button. 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! **NOTE: Installation will erase all existing data in a pre-existing database. When developing your profile always work with an empty CollectiveAccess database so you can continually reload your profile for testing.**

The Installer will notify you if your profile has installed correctly or if there were errors. If it installed correctly, login to your site and test your system. If the installer reports some errors, you will have to find and make those corrections. **Please refer to the CollectiveAccess wiki for further descriptions of error messages in the Installation Error Message Dictionary.**

You can continue to make and save changes to your profile as long as you are in the testing phase of your system. To overwrite an installed profile, you select the overwrite button on the installation webpage. This will replace the old profile with the newest version; however it will **erase** any data that was entered in the system. So be careful not to enter important information while testing your profile.

## 6.0 Conclusion

Congratulations! The profile is finished, testing is complete, and you're about to begin using your new Collection Management System by CollectiveAccess. Please refer to the next manual in this series, **Cataloging in CollectiveAccess for Beginners** to get a basic grounding on cataloging principles and best practices.

Don't hesitate to contact us if you have any questions at:  
[support@collectiveaccess.org](mailto:support@collectiveaccess.org)

Also keep up to date with the latest developments and changes through our forums and wiki. Feel free to join in the discussion!

## **7.0 Resources**

<http://wiki.collectiveaccess.org/>

<http://www.dublincore.org/>

<http://www.loc.gov/standards/>