

# NUL DIGITAL IMAGE LIBRARY VRA CORE METADATA EDITOR TECHNICAL DOCUMENTATION

---

*Bill Parod*  
*March, 2012*

<b>INTRODUCTION.....</b>	<b>2</b>
<b>VRA EDITOR .....</b>	<b>2</b>
XML INSTANCES.....	2
<i>Editable Instances .....</i>	<i>3</i>
<i>VRA Templates.....</i>	<i>3</i>
<i>Select Lists.....</i>	<i>3</i>
<i>Configuration .....</i>	<i>4</i>
<i>Application State .....</i>	<i>4</i>
BIND STATEMENTS .....	5
SUBMISSIONS.....	5
EVENT HANDLERS.....	6
XBL INCLUDES.....	7
CSS.....	7
MAIN HEADING.....	8
VRA WORK SET TABS.....	9
<i>Adding and Deleting Sets .....</i>	<i>9</i>
<i>VRA Display Field.....</i>	<i>10</i>
<i>Repeated Fields.....</i>	<i>10</i>
<i>XBL Sets.....</i>	<i>10</i>
VRA RELATIONSHIPS TAB.....	11
VRA IMAGE TAB.....	11
ACTION BUTTONS.....	12
DIALOGS .....	12
<b>SEARCHING .....</b>	<b>13</b>
<b>AUTHORITIES CLIENT .....</b>	<b>14</b>
AUTHORITY CONTROLLED DESCRIPTIVE FIELDS .....	15
AUTHORITIES CLIENT XBL.....	15
XML INSTANCES.....	17
<i>Editable Instances .....</i>	<i>17</i>
<i>Authority Record Templates.....</i>	<i>17</i>
<i>Select Lists.....</i>	<i>18</i>
<i>Configuration .....</i>	<i>18</i>
<i>Application State .....</i>	<i>19</i>
BIND STATEMENTS .....	19
SUBMISSIONS.....	19

AUTHORITIES LOOKUP XBL .....	19
AUTHORITIES CREATION .....	20
<i>Heading Types</i> .....	21
<i>Personal Heading XBL</i> .....	23
<i>Corporate Heading XBL</i> .....	23
<b>DEPLOYMENT (FILE) STRUCTURE</b> .....	<b>23</b>
EDITOR .....	23
XBL COMPONENTS .....	24
<b>DEPENDENCIES</b> .....	<b>25</b>
FEDORA .....	25
<i>Fedora Image Disseminator</i> .....	25
<i>Fedora VRA Access</i> .....	25
<i>Fedora SVG Access</i> .....	25
IMAGE SERVER TILE INTERFACE .....	25
AUTHORITIES SERVICE .....	25
DIL HYDRA REST API .....	25
<i>Search</i> .....	25
<i>Save</i> .....	25
<i>Clone</i> .....	25
<i>Delete</i> .....	25
<i>Display</i> .....	26

## Introduction

This document describes the implementation of the Digital Image Library VRA Core Metadata Editor. The Editor is an XForms application running on the Orbeon community release web application. It was developed and currently runs on Orbeon Forms 3.8.0.201005270113 CE.

The Editor is comprised of several components. These include the main form (*WEB-INF/resources/apps/vra/forms/editor.xhtml*), and several XBL components that encapsulate discrete and reusable structures or functionality in the Editor. This document describes these components, their constituent parts, and how they work together.

## VRA Editor

This section describes the various code sections within *editor.xhtml*.

### XML Instances

XForms applications refer to the XML documents they edit as well as all XML data structures used within the application with XForms *instances*. This section describes the XML instances declared within the VRA Editor.

## Editable Instances

These are the two XML model instances that the editor exposes to editing.

Instance ID	Description
vra-work-instance	The VRA Work document being edited
vra-image-instance	The VRA Image document being edited

## VRA Templates

The editor supports addition and deletion of VRA sets as well as unlimited inserts of repeatable elements. The XML templates that represent specific parts of the VRA and back their interface widgets in the editor are referenced as instances:

Template Instance ID	VRA Element
titleset-instance	vra:titleSet
descriptionset-instance	vra:descriptionSet
inscriptionset-instance	vra:inscriptionSet
agentset-instance	vra:agentSet
dateset-instance	vra:dataSet
styleperiodset-instance	vra:stylePeriodSet
worktypeset-instance	vra:workTypeSet
worktype-instance	vra:workType
materialset-instance	vra:materialSet
measurementsset-instance	vra:measurementSet
culturalcontextset-instance	vra:culturalContextSet
techniqueset-instance	vra:techniqSet
technique-instance	vra:technique
locationset-instance	vra:locationSet
location-instance	vra:location
sourceset-instance	vra:sourceSet
source-instance	vra:source
subjectset-instance	vra:subjectSet
relationset-instance	vra:relationSet
relation-instance	vra:relation

## Select Lists

All the various select lists in the editor, whether for feature control (e.g. sort key) or VRA controlled terms (e.g. "pref" yes/no) are kept in XML files and referenced by the application in instances. Each selection has a label displayed to the user and an internal value that is inserted into the edited document when selected.

Select List ID	Select list use
relationtype-select	Work/Image relationship

relationpref-select	Indicate if the relationship is “preferred”
locationtype-select	Type of Location
bool-select	True/False for boolean attributes
roletype-select	Agent roles *** replaced by attribution-type
attributiontype-select	Agent attributions
cultures-select	Agent culture
worktype-select	Worktype

### Configuration

The Editor integrates directly with a variety of external services. Service connect information in the form of URL templates and other settings are abstracted out of the application into a configuration file expressed in XML. Configuration settings include:

<b>Fedora URL Segments</b>
Fedora server domain name and PATHINFO string
Fedora suffix (after provided PID) to obtain the VRA datastream
Fedora suffix (after provided PID) to obtain the DELIV-OPS datastream
Fedora suffix (after provided PID) to obtain a thumbnail image
Fedora suffix (after provided PID) to obtain an html page
<b>Hydra and Fedora Management Services</b>
URL to service for cloning a Fedora object with provided PID
Hydra service URL for updating Fedora VRA datastream with the current VRA Image or Work instance
Hydra service URL for deleting a Fedora object
Hydra service URL for cloning a Fedora object
Hydra service URL for Searching DIL
Hydra service URL for displaying an image page
Hydra Query for Searching DIL
<b>Tile Image Viewing URL Segments</b>
Zooming Viewer URL
Tile Server URL
<b>PSDS Management Services</b>
URL to notify PSDS that the current item's cataloging is complete
<b>Authorities Service</b>
Authorities service Domain Name

### Application State

Internal data structures for managing application state are also managed as XML. These include:

Search results obtained when searching DIL.
Temporary VRA documents retrieved to probe related Images and Works.

State information for various pop-up dialogs
State information indicating enabled/disabled state of application buttons.
Image and Work 'dirty' states
Instances for service call replies
Current Image Object PID
Current Image Object title
Current Work Object PID
Current Work Object title
Current Work's Preferred Image PID
Current PSDS Item associated with the current image
Image and Work Drop well information

## Bind Statements

We use XForms *bind* statements to manage *readonly* status of various buttons based on conditions in the document expressed. For example, we enable/disable the "Add Agent" button based on whether there is already a <vra:agentSet/> in the Work or Image. We don't list all the statements in this section of the editor here, but they are all of this form:

```
<xforms:bind nodeset="//buttons:add_agentset" readonly="count(instance('vra-work-instance')/vra:work/vra:agentSet)=1"/>
```

```
<xforms:bind nodeset="//buttons:remove_agentset"
readonly="count(instance('vra-work-instance')/vra:work/vra:agentSet)=0"/>
```

## Submissions

XForms *submission* elements are used to invoke external network services and optionally replacing internal XML structures with service replies. The VRA Editor includes several such *submissions* to interact with our various collection management services:

Load a VRA Image instance for editing
Load a VRA Work instance for editing
Load a VRA Image instance temporarily to obtain related information from it, such as related Works.
Load a VRA Work instance temporarily to obtain related information from it, such as related Items.
Submit a clone request, loading the resulting VRA datastream in the "clone_reply" instance.
Get an image object's SVG datastream.
Search DIL

Submit edited Work
Submit edited Image
Load empty Work template
Load empty Image template
Reinitialize Search template
Delete VRA Work in Hydra
Clone VRA Work in Hydra
Submit PSDS 'Cataloged' Notification

## Event Handlers

XForms employs an event model to register and trigger handlers for a variety of events, such as when a document is finished loading, a window is closed, a form value is changed, etc... Event handlers are frequently scoped within a specific interface widget, for example to provide special handling for data entry for that widget. Handlers can also be declared more broadly and used to coordinate behavior across the application. The VRA Editor provides handlers for some XForms specific events but also declares several of its own internal events to facilitate flexible handling of application behavior and consolidate common logic in associated handlers. Event dispatch and handling is also used to bridge XForms and Javascript contexts, as in Drag and Drop with jQuery. This code section includes all the application-scoped event handlers in the Editor. These include:

Handle any change event to the work instance
Handle any change event to the image instance
This action is dispatched to append *Set field values to their *Set/vra:display field
This event is dispatched to attach cloning when dragging to draggable objects when new draggable objects are added to the application, such as when a search is performed or a relation is added to a work or image. The function here that is added to draggable objects, obtains PID, Type, and Title from the dropped div and points them into hidden inputs (e.g. editDroppedImage-pid) which are referenced by elements in the "request_params" instance of the main Xforms model. XForms events are then dispatched to obtain the dropped PID from the model and load/update related model instances.
This action is called from javascript when an image is dropped on a Image or Work well signifying the user wants to edit the Image/Work
This action is called from javascript when a work is dropped on a Image or Work well signifying the user wants to edit the Work
load the work in "instance('request_params')/rp:work_pid", set up by the above Javascript function
Get the Work's related image and set for Work's thumbnail in the Form heading.
Update currently edited Image
Clear the dirty flag for Work
This action is called from javascript to add a image relationship to the current work
add drag capability to relations

This action is called from javascript to add a work relationship to the current image
Read image vra and parse out the work id
Add drag capability to relations
The user has dropped an Image (from search results) into the Work well. We want to add this Image to the current Work's image relations.
The user has dropped a Work (from search results) into the Work well. We want to add this work's preferred image to the current Work's image relations.
Reinitialize the request_params instance.

### XBL Includes

XForms uses the W3C XML Binding Language (XBL) to encapsulate XForms functionality in separate components and instantiate those components in a form using custom XBL-bound XML elements. The VRA Editor makes use of this mechanism to encapsulate repeated or auxiliary functionality in the application. The next section of the editor includes these XBLs for reference within the application. The following XBLs are included relative to the orbeon webapp's *WEB-INF/resources/xbl* directory:

<b>Authorities Tool Client XBLs</b>
nul/authoritiesClient/authoritiesClient.xbl
nul/authoritiesLookup/authoritiesLookup.xbl
nul/authoritiesPersonalHeading/authoritiesPersonalHeading.xbl
nul/authoritiesCorporateHeading/authoritiesCorporateHeading.xbl
<b>VRA Set editor XBLs</b>
nul/vraSet/vraSet-agentSet.xbl
nul/vraSet/vraSet-titleSet.xbl
nul/vraSet/vraSet-descriptionSet.xbl
nul/vraSet/vraSet-inscriptionSet.xbl
nul/vraSet/vraSet-dateSet.xbl
nul/vraSet/vraSet-stylePeriodSet.xbl
nul/vraSet/vraSet-culturalContextSet.xbl
nul/vraSet/vraSet-materialSet.xbl
nul/vraSet/vraSet-measurementsSet.xbl
nul/vraSet/vraSet-subjectSet.xbl
<b>Search and DragNDrop XBLs</b>
nul/vraSearch/vraSearch-results.xbl *** Not in use; Should be removed
nul/vraWork/vraWork-thumbnail.xbl

### CSS

The next section of the editor includes CSS for overriding XForms CSS classes and adding additional ones for our custom HTML. One reason the editor has the “look and feel” of a metadata “editor” as opposed to a “form” is in its layout and

appearance. Its appearance especially, deviates from the stock XForms appearance Orbeon provides. This is achieved through CSS. The process of identifying affecting Xforms classes to override in situations, as well as appropriate scope to leverage in new CSS is painstaking and more emergent than proscriptive. Currently, CSS is included in a separate css file, found inline within a <xhtmlstyle/> section in the form's header, and also in many cases found within "style" attributes on individual elements. A final consolidation and refactoring process is needed to complete the CSS work. For now the majority of CSS is found in the mentioned <style/> element in this section of the editor.

## Main Heading

The VRA Editor is designed to present VRA Works and Images individually and in relation to each other. The editor supports simultaneous editing of a VRA Work and VRA Image, loading each in their respective parts of the editor. When a Work is selected from search results for editing, its 'preferred' image is also loaded into the editor. The converse is true when an Image is selected for editing – its Work is loaded into the editor as well. The Main Heading runs across the top of the editor and displays the thumbnail image and brief metadata for the Work (on the left) and Image (on the right) in the editor. Their respective thumbnail areas also serve as "drop targets". Thumbnails presented in search results and in VRA Relations can be dragged and dropped into these "drop targets". When dropped without holding any keys down, the drop causes the editor to load the dropped item for editing. The dropped item, whether Work or Image and whether dropped on the Work or Image thumbnail area, will cause the associated preferred Image and Work to be loaded into Work and Image areas of the editor. If the user holds down the option/alt key while dragging an Image to the Work drop target, that Image will be added as a VRA *imageIs* Relation to the currently edited Work. If the user holds down the option/alt key while dragging a Work to the Image drop target, that Work will be added as a VRA *imageOf* Relation to the currently edited Image.



VRA Work Drop Target

VRA Image Drop Target

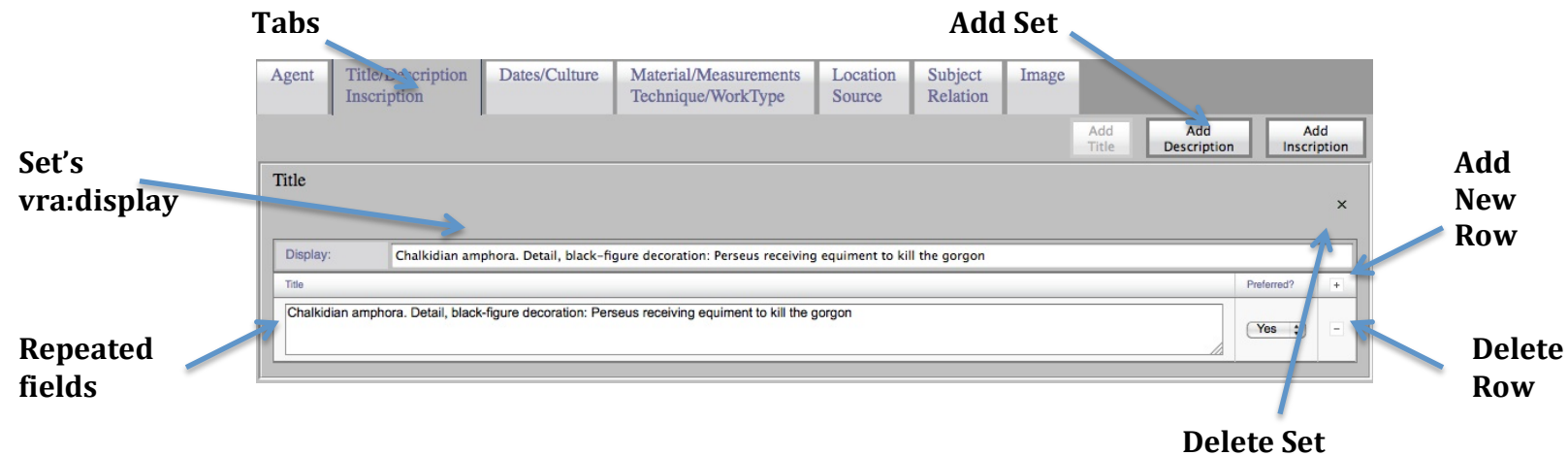


Drag and Drop in the editor is implemented with the *jQuery UI* Javascript library. XHTML DIV elements declaring the *imageWell* class are enhanced with the *jQuery UI droppable* Javascript callback. This callback is invoked on a drop and saves item identifier (Fedora PID) and title in XForms instance data and dispatches a custom XForms event handler which loads the associated VRA objects' metadata in the editor. The Main Header serves to display the currently editing Work and Image and also function as an active drop area for loading other items.

## VRA Work Set Tabs

VRA Core organizes its full descriptive model into “sets”, such as titleSet, agentSet, dateSet, descriptionSet, subjectSet, etc. Each set contains repeatable groups of fields, some using controlled vocabularies, and a free-form ‘display’ field that is used to combine or summarize the set for public display purposes.

The Editor's presents these Sets in a *tabview*, laying out the various VRA Sets in a set of *tabbed panels*. The VRA Work is supported by a full set of VRA Sets, organized into six tabbed panels. A seventh panel contains VRA Set widgets for the VRA Image.



## Adding and Deleting Sets

Each panel offers specific Sets of the VRA Core for editing. In the example above, one can edit the *titleSet*, *descriptionSet*, and *inscriptionSet*. The Editor is configured to offer editor widgets for some sets by default and others optionally. In this example, the *titleSet* is available by default. Since \*Sets are not repeatable in VRA Core, if the current item already includes a specific Set, its “Add” button is disabled, as “Add Title” is above. Since in this example does not already include *descriptionSet* or *inscriptionSet*, their “Add” buttons are enabled. You can also delete an entire Set by clicking on its “X” delete button on its upper right.

### VRA Display Field

Each Set contains a single *vra:display* field. This field accumulates repeated fields in the Set for a summary field displayed to users. It also under complete control of the cataloger who is free to adjust the field as needed for human readers.

### Repeated Fields

In addition to a single *display* field, each Set usually includes repeatable aggregations of descriptive fields. In the *titleSet* example above, the repeated field aggregation is a *title* and an indication if it is the *preferred* title. Some Sets include more fields in their repeatable aggregates, other include just a single field.

In XForms we implement repeatability with a table structure, each row containing the set of fields in the repeated aggregate. One can add new rows with the small *plus* at the upper right of the table. Any row can be deleted with the *minus* to its right.

### XBL Sets

Internally within Editor, each *tabbed panel* contains a structure with buttons to add its various Sets and below that, a structure for each Set containing its name, delete button, and reference to an XBL that implements the detailed editing of the Set. These XBL-encapsulated editor widgets are listed in the “VRA Set editor XBLs” section above. Since XBLs are instantiated with simple XML defined by their binding, this arrangement reduces the amount of code directly in the editor, allow for reuse (such as in both Work and Image editing), and supports more isolated maintenance. To include the *titleSet* XBL in the above example, the Editor provides the following XML:

```
<nulVraSet:titleSet ref="."/>
```

**nulVraSet** is the custom namespace declared for our Set XBLs. **titleSet** is the specific XBL. **ref="."** binds the current XML context node to the context in scope within the XBL. In the above case, the current XML context is the *vra:itemSet* for the Work we’re editing.





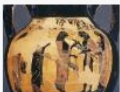

This same arrangement is used for the most of the VRA Sets within the Work and Image Editors. The full list of VRA Set XBLs (*WEB-INF/resources/xbl/vraSet*) includes:

VRA Set XBL Components
vraSet-agentSet.xbl
vraSet-culturalContextSet.xbl
vraSet-dateSet.xbl
vraSet-descriptionSet.xbl
vraSet-inscriptionSet.xbl
vraSet-materialSet.xbl

vraSet-measurementsSet.xbl
vraSet-stylePeriodSet.xbl
vraSet-subjectSet.xbl
vraSet-titleSet.xbl

## VRA Relationships Tab

Managing relationships between Works and Images is key feature as well as significant challenge of VRA Core. The Relationship tab for Works and the Relationship widget for Images also employ a structure like the other Sets described above. That is, it contains a *vra:display* field and a table structure aggregating relationship information in each row. But relationships aren't are not added within the relationship editor itself. Instead, relationships are added by dragging Works or Images from search results into their opposite drop target thumbnail as described in the “Main Heading” section above. The result looks like this:

Relations				
Display: Chalkidian amphora. Detail, black-figure decoration: Perseus receiving equipment to kill the gorgon; Black-figure cup. Inside: two bodies, one head fig...; Black				
Work	Relation	Image	Preferred?	
 Chalkidian amphora. Detail, black-figure decoration: Perseus receiving equipment to kill the gorgon	Has related Image: ▾	 Chalkidian amphora. Detail, black-figure decoratio...	Yes ▾	-
 Chalkidian amphora. Detail, black-figure decoration: Perseus receiving equipment to kill the gorgon	Has related Image: ▾	 Black figure Panathenaic prize amphora. Athena Pro...	No ▾	-
 Chalkidian amphora. Detail, black-figure decoration: Perseus receiving equipment to kill the gorgon	Has related Image: ▾	 Black-figure cup. Inside: two bodies, one head fig...	No ▾	-

The Relations Tab is also different from other Editor tabs in that it references other objects that what it is editing. It also applies conditional logic in what it presents. For example, to display a thumbnail for a Work, the Editor examines if the Work has a *preferred* image relationship. If not it simply uses the first of its *imageIs* relations to obtain an Image identifier. It then forms a thumbnail URL with that identifier and configured URL prefixes and suffixes to form a Fedora thumbnail dissemination.

## VRA Image Tab

The VRA Image Tab packs a zooming viewer and its VRA Sets within a single *tab panel*. The Image Editor however does not include all VRA Sets the Work editor does. It includes only *titleSet*, *agentSet*, *dateSet*, *descriptionSet*, and *subjectSet*.

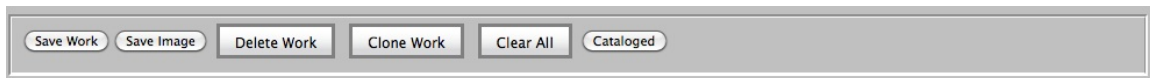
The zooming viewer is currently the Flash-based viewer. This viewer obtains image tiles directly from the Aware Tile Server using its REST API as needed per the users

zooming and panning. The viewer URL and Aware Tile Server URL are provided in configuration settings (“*Configuration*” section above). The image file path is obtained from the Image’s SVG (Fedora Datastream ID DELIV-OPS) file. These are combined in XHTML *object* and *embed* elements.

The *titleSet*, *agentSet*, *dateSet*, *descriptionSet*, and *subjectSet* editors use the same structure and XBLs as described above in “*XBL Sets*”.

## Action Buttons

Below the main Editor form is a set of buttons for contacting external services.



The *Save Work* and *Save Image* buttons invoke *submissions* which submit their XML (Work or Image respectively) to Hydra. Hydra in turn updates their associated Fedora objects and updates its Solr index.

*Delete Work* dispatches a custom handler which invokes a confirmation dialog. On confirmation, the dialog submits a delete request to Hydra for the Work, and reinitializes the current Work and Image XML instances.

*Clone Work* similarly dispatches a custom handler which invokes a confirmation dialog. On confirmation, the dialog submits a clone request to Hydra. Hydra returns a new PID for the cloned Work. The dialog then replaces the Works identifier (vra:work/@refid), load the new Work’s VRA, clear the Image VRA, and the Image viewer.

*Clear All* clears Editor state, including VRA Work XML, VRA Image XML, search results, search query parameters, current editor identifiers, and the SVG image for the zooming viewer.

The *Cataloged* button is available when the Editor is invoked from PSDS, passing in an item’s PSDS identifier. When a workflow item identifier is available to the editor, the cataloger can use *Cataloged* to notify the workflow system when the item’s cataloging is complete. The workflow system can then advance the item if cataloging was the only process pending for its publication or fulfillment. This way a “save” action isn’t overloaded with both saving and indicating that the item is ready. Catalogers may catalog (and save) an item over multiple edit sessions before it is deemed ready for release.

## Dialogs

Following the *Action Buttons* in the Editor are various dialogs. These include those presented when a user clicks on a search result (described in the next section) and all the confirmation dialogs presented when the user requests any change to objects in the repository or other potentially destructive actions.

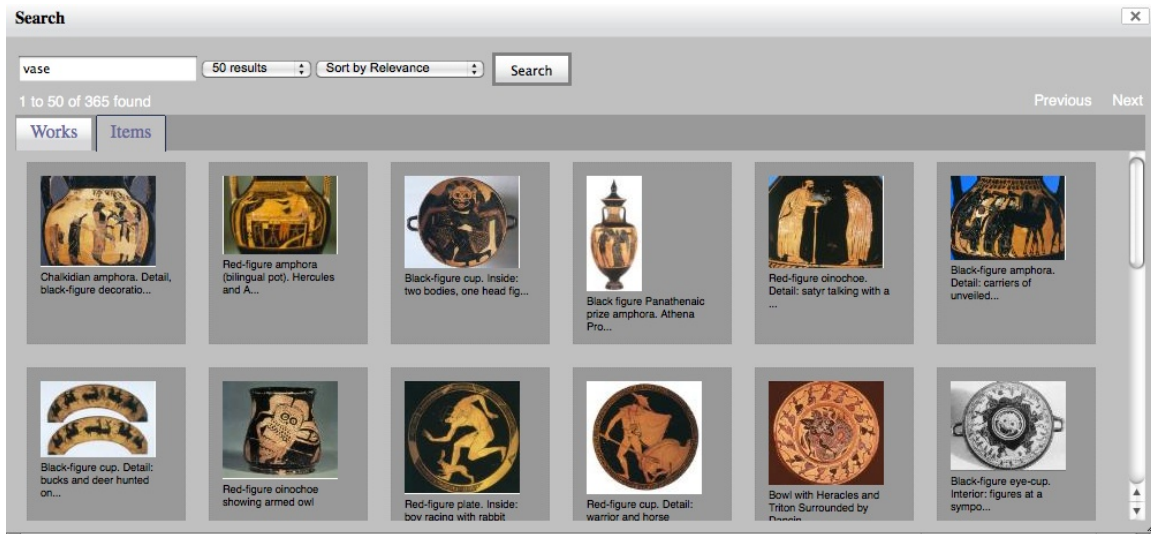
## Searching

The Editor integrates searching of the Digital Image Library. The search interface is presented in a pop-up dialog when clicking on the “*Search*” link in the upper right corner of the Editor. The interface and code structure of the search interface is similar to that of that Editor. It includes a heading area and a set of tabbed panels.

In this case the heading includes controls for search parameters, result set count, and *previous* and *next* buttons. The *tabview* includes a *tab* for Work search results and one for Image search results. The popup is resizable. Search results can be dragged to drop targets in the Editor as described above in the “*Main Heading*” section.

Search parameters are maintained in an internal XML instance (*search-query*). These include the search term, sort key, the result set page number, and requested page size. These values are combined with a configured endpoint URL – in our case, a Hydra URL to form a complete search request URL. The reply to this search request is maintained in another internal XML instance (*dilsearch-instance*).

We iterate through the search results, once for Works and once for Image, segregating each in their own *tab panel*. Each search result is wrapped in an XHTML DIV, the item’s identifier and title attached as DIV attributes. These DIVs are also enhanced with “clone when dragging” properties using jQuery UI. This allows them to be dragged to Editor drop targets. This enhancement is performed with a custom event (*listing-init-drag* ) dispatched after each search. An associated “drop” uses the DIV identifier and title attributes to reference the dropped item for loading in the editor or asserting a relationship.



The search interface also facilitates various useful operations that draw on the larger collection. The editor supports the following actions as pop-up actions on search results:

- Add this Image to the current Work*
- Edit this Image record*
- Edit this Image's Work record*
- Edit this Work record*
- Clone this Work and edit it*
- Display this Image in a separate window*

These actions are presented in dialogs invoked when clicking on a search result. There is a dialog for Work search results and a separate dialog for Image search results. Each dialog includes event handlers that implement their actions in response to a selection event (DOMActivate) on each action link (*xforms:trigger*). This is another example of the use of the event model for application behavior.

## Authorities Client

Since controlled vocabularies are used for so many fields, there's a strong incentive to make authority selection easy to do throughout the editor. We accomplish this by first encapsulating an authorities client in an XForms dialog. This dialog implements the authorities lookup interface and authorities service interactions in a distinct XForms component. There also needs to be a way to associate the encapsulated dialog with controlled fields within the editor. That is, to designate certain descriptive fields to be under authority control and invoke the authorities dialog when controlled fields are edited. Orbeon supports the W3C XML Binding Language (XBL) for binding an XForms MVC component (the authorities dialog in this case) with a custom namespaced element. By using XBL to create a custom XML element



for controlled field input, invoking the authorities dialog when entered, that element can easily be used for any field under authority control in the Editor.


### Authority Controlled Descriptive Fields

An example of this is the *vra:agentSet/vra:agent/vra:name* element. Within the main Editor form, we include an XBL for editing the *vra:agentSet*:

```
<nulVraSet:agentSet ref="." authorityToolStyle="dialog"/>
```

**nulVraSet** is the custom namespace declared for our Set XBLs. **agentSet** is the specific XBL. **ref="."** binds the current XML context node to the context in scope within the XBL. In the above case, the current XML context is the *vra:agentSet* for the Work we're editing. **authorityToolStyle="dialog"** indicates to the **agentSet** XBL that we want to use our authorities dialog for controlling this field. The only option currently supported for this setting is **dialog**. The setting is present though in anticipation of an **inline** option, supporting autocomplete lookup directly within Editor descriptive fields. XBL component factoring in the application will support this, in that lookup functionality is encapsulated in its own XBL. This example of using an XBL component to encapsulate editing of a VRA descriptive field is much like that discussed above for *vra:titleSet* in the "XBL Set" section. However in this case (*vra:agentSet*), our Editor XBL component (**nulVraSet:agentSet**) uses another XBL component, the authorities client XBL, for one of its fields.

The **nulVraSet:nulVraSet** XBL (below) uses **nulAuthCl:authoritiesClient** for each agent's name (*vra:agentSet/vra:agent/vra:name*):



Sort Name	Early Date	Late Date	Culture	Attribution	Extent	Refid	
Kandinsky, Vassily (Russian painter	1866	1944		-- Select One --		aat 500021090	-
Jawlensky, Alexei (Russian painter,	1864	1941		-- Select One --		aat 500017981	-

```
<nulAuthCl:authoritiesClient fieldType="vraname" ref="vra:name"/>
```

### Authorities Client XBL

The XBL implementation bound to **nulAuthCl:authoritiesClient** is found in the orbeon webapp in *WEB-INF/resources/xbl/nul/authoritiesClient*. It encapsulates interactions with the Authorities Service to support forms for browsing and creating name authorities. It appears as a simple input field, as in the above example. However when that input field achieves focus (as when a user clicks within it), this XBL shows the Authorities Client dialog panel.

**Authorities Service**

**Browse Headings**

kandinsky,

**Kandinsky, 1866-1944**

Search under: Kandinsky, Wassily, 1866-1944

Kandinsky, Alexei Ivanovich

Search under: Kandinskiĭ, A. I.

**Create a New Heading**

-- Choose a Heading Schema --

This dialog includes an authority heading browser with autocomplete that allows the user to search headings by typing in text. When the heading they are looking for is found, they select it and then apply it to the descriptive field they are editing with the “Apply” button in the Authorities Client panel.

**Authorities Service**

**Browse Headings**

Kandinsky, Wassily, 1866-1944

Apply or Cancel

**Create a New Heading**

-- Choose a Heading Schema --

The heading is then applied to the descriptive field, inserting its authorized form in the main field, its record number in the refid, and if the record is for a personal name, its start and end dates in the agent start and end dates, as in the example below:

Agent Display Kandinsky, Wassily, 1866-1944							
Sort Name	Early Date	Late Date	Culture	Attribution	Extent	Refid	+
Kandinsky, Wassily, 1866-1944	1866	1944		-- Select One --		lcnao N7905931v	

This XBL includes all its own XML state instances, edit instances, select instances, event handlers, submission statements, forms, and bind statements. In other words,



it very much represents a complete and separate form application. These are described in the sections below.

### XML Instances

The Authorities Client has fewer XML instances than the VRA Editor, but enough that they're worth documenting here.

### Editable Instances

There is one XML model instance that the Authorities Client creates for use outside itself, a formulateAuthority XML structure for creating new records in the Authorities Service. Its structure is described in "nulAuth: schemas for the formulation or creation of an authority record from raw data" by Gary Strawn.

Instance ID	Description
authority-create-record	Authority record submitted for record creation

### Authority Record Templates

The client can create a variety of headings types. These may be repeated and may in turn include optional non-heading parts. There are a lot of possible fields (XML structures) that may be included when creating a new authority record. The client uses a set of XML template files to express all these structures and insert them as needed into the new authority record. The XML templates that represent specific parts of the authority record and back their interface widgets in the client are referenced as instances. Their Ids are listed here and a self-explanatory regarding what they contain:

Template Instance ID
lcnaco-nameCorporate-preferred-instance
lcnaco-nameCorporate-related-instance
lcnaco-nameCorporate-variant-instance
lcnaco-namePersonalSimple-preferred-instance
lcnaco-namePersonalSimple-related-instance
lcnaco-namePersonalSimple-variant-instance
ulan-nameCorporate-preferred-instance
ulan-nameCorporate-related-instance
ulan-nameCorporate-variant-instance
ulan-namePersonalSimple-preferred-instance
ulan-namePersonalSimple-related-instance
ulan-namePersonalSimple-variant-instance
non-heading-dates-instance
non-heading-date-instance
non-heading-roles-instance
non-heading-role-instance

non-heading-places-instance
non-heading-place-instance
non-heading-nationalities-instance
non-heading-nationality-instance
non-heading-genders-instance
non-heading-gender-instance
non-heading-languages-instance
non-heading-language-instance
non-heading-notes-instance
non-heading-note-instance
non-heading-note-noteText-instance
non-heading-source-informationFromSource-instance
non-heading-sources-instance
non-heading-source-instance
non-heading-codedElements-instance
non-heading-codedElement-instance
non-heading-workAspects-instance
non-heading-workAspect-instance
non-heading-familyData-instance
non-heading-familyInformation-instance
non-heading-unestablished-part-instance

### Select Lists

All the various select lists used in the Authorities Client are kept in XML files and referenced by the XBL in instances. Each selection has a label displayed to the user and an internal value that is inserted into the edited document when selected.

Select List ID	Select list use
authorities-select	Available heading choices: ULAN or LC
non-heading-elements	Available non-heading elements
establishedPart-type	Available established part types
non-heading-codedElement-name	Available coded element names
non-heading-date-type	Available date types
non-heading-note-type	Available note types
non-heading-role-type	Available role types
non-heading-workAspect-type	Available work aspect types
non-heading-gender-gender	Available genders
unestablishedPart-personal-type	Available unestablished part types

### Configuration

There is only one configuration parameter obtained from the main Editor configuration file. It is the main URL to the Authorities Service:

Authorities Service
Authorities service IP

### Application State

Internal data structures for managing application state are also managed as XML. These include:

Application State Instances	Purpose
valid-heading	Control enabled/disabled for Apply button
valid-new-heading	Control enabled/disabled for Create button
record-key	Return value for Create Record request
authority-fullrecord	Full authority record requested for browse
authorized-termTypes	Legal heading types for the requested schema

### Bind Statements

As within the Editor we use XForms *bind* statements to manage *readonly* status of various buttons and checkboxes based on conditions in the authority record being created.

### Submissions

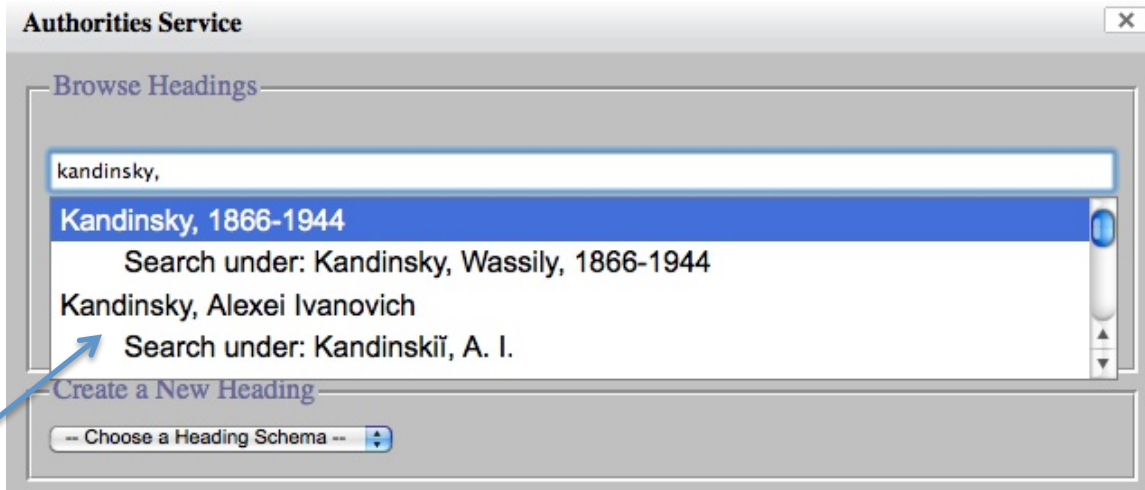
XForms *submission* elements are used to invoke external network services and optionally replacing internal XML structures with service replies. The VRA Editor includes such *submissions* to interact with our various collection management services:

Create an authority record by submitting an XML record to the Authorities Service
Reinitialize the Authority Create Record
Reinitialize the selected authority type

### Authorities Lookup XBL

The autocomplete input used in the authority browser above is implemented as a separate XBL. It is called **nulAuthC1:authoritiesLookup** and included in the above panel with:

```
<nulAuthC1:authoritiesLookup fieldtype="vraname"
    headingtype="all" bind="autocomplete-bind"/>
```

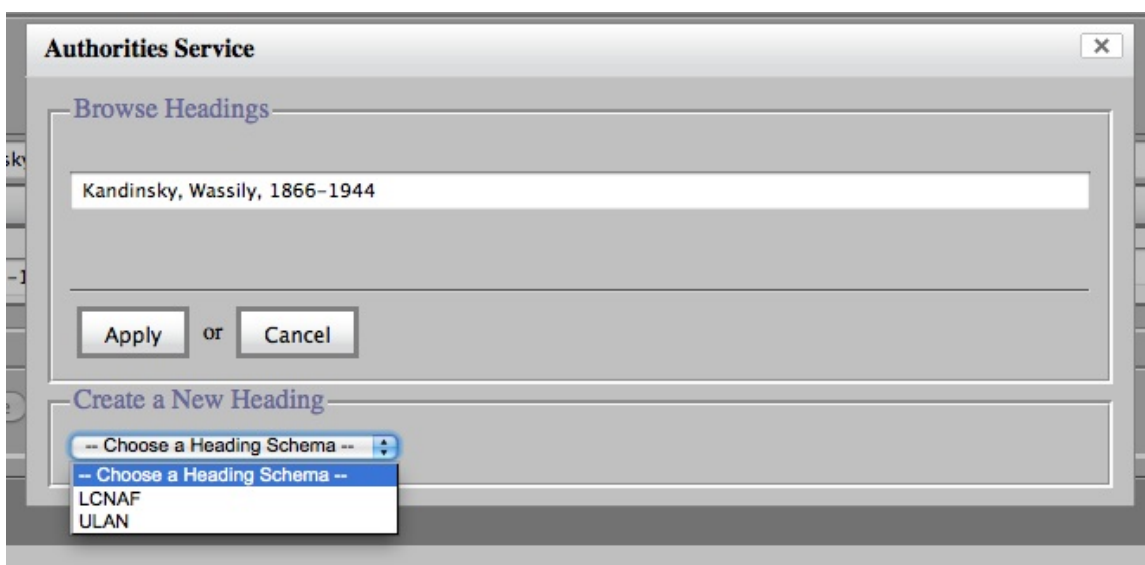


```
<nulAuthC1:authoritiesLookup .../>
```

This XBL encapsulates the autocomplete behavior and all the query interactions with the Authorities Server as the user types characters in its input form. Because this lookup functionality is its own XBL component, it can be used in other parts of the application, such as in a descriptive field for **inline** autocomplete. Its **fieldtype** parameter value of **vraname** indicates it is being used for a VRA name field and so should apply the authority record's birth and death dates to the VRA record. This functionality is encapsulated in the **authoritiesLookup** XBL.

### Authorities Creation

If the cataloger doesn't find the heading they need, they may create a new one. This is done by selecting the desired authority schema from a drop-down list under "Create a New Heading":



## Heading Types

Once the user has selected their desired schema (currently LCNAF and ULAN), the panel expands to allow them to select the type of heading they wish to create, (currently Simple Personal Name or Corporate Name):

The screenshot shows a window titled "Authorities Service" with a close button in the top right corner. The window is divided into two main sections. The top section, "Browse Headings", contains a text input field with the text "Kandinsky, Wassily, 1866-1944" and two buttons, "Apply" and "Cancel", separated by the word "or". The bottom section, "Create a New Heading", features a dropdown menu currently set to "LCNAF". Below this is a row of six tabs: "Preferred", "Variant", "Related", "Other", "Debug", and "Debug Reply". The "Preferred" tab is selected. A dropdown menu is open below the "Preferred" tab, showing the text "-- Choose a Heading Type --" followed by "Simple Personal Name" and "Corporate Name". At the bottom of this section are two buttons, "Submit Your New Authority" and "Cancel", separated by the word "or".

The panel then expands to offer a set of tabbed panels for adding a Preferred Heading, multiple Variant Headings, multiple Related Headings, and Other non-heading parts to the authority records they're creating:

**Authorities Service** [X]

---

**Browse Headings**

Kandinsky, Wassily, 1866-1944

Apply or Cancel

---

**Create a New Heading**

LCNAF

Preferred Variant Related Other Debug Debug Reply

Simple Personal Name

**Preferred Personal Heading**

☒ Individual Fields ☐ Established/Unestablished Parts

Surname:

Fuller Surname:

Abbreviated Surname:

Forename:

Fuller Forename:

Abbreviated Forename:

☐ Begins with Forename ☒ Unique Name

Associated Title Phrase:

Place Name:

☐ Entry Element

Associated numbering:

Submit Your New Authority or Cancel

<nulAuthC1:authoritiesPersonalHeading ref="."/>

Since Personal and Corporate Headings are repeated throughout the Authorities Creation forms, they have been implemented as reusable XBL components. These XBLs encapsulate their inputs and specific business rules for managing their respective fields.

### Personal Heading XBL

The Personal Heading XBL is found in /WEB-INF/resources/xbl/nul/authoritiesPersonalHeading. It can be instantiated in other forms, such as the Authorities Creation form as follows:

```
<nulAuthCl:authoritiesPersonalHeading ref="."/>
```

### Corporate Heading XBL

The Corporate Heading XBL is found in /WEB-INF/resources/xbl/nul/authoritiesCorporateHeading. It can be instantiated in other forms, such as the Authorities Creation form as follows:

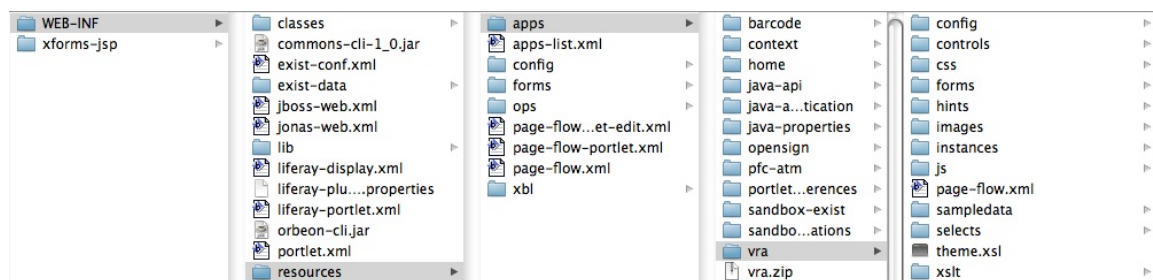
```
<nulAuthCl:authoritiesCorporateHeading ref="."/>
```

## Deployment (file) Structure

The VRA Editor and associated XBL components are deployed within the Orbeon Web Application running in a Tomcat Servlet Container. Web applications, including *orbeon* are installed in Tomcat's 'webapps' directory. Beneath *orbeon* in the webapps directory are the following directory structures:

### Editor

The Editor application files are under /WEB-INF/resources/apps/vra:



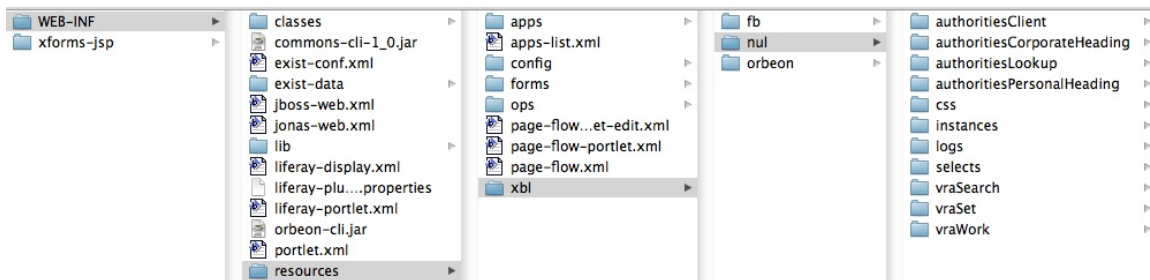
The directory structure includes directories for:

Directory	Purpose
config	Contains configuration file for VRA Editor
controls	No longer in use
css	Contains CSS file for VRA Editor
forms	Contains the main form, <i>editor.xhtml</i>
hints	Tooltip hints for Editor fields

images	Contains application icons such as plus and minus
instances	Contains template instances
js	Contains jQuery UI Javascript file for VRA Editor
page-flow.xml	Orbeon file for routing URLs to form files
sampladata	Sample XML used for testing
selects	XML instances for select lists
theme.xsl	Orbeon file for form processing
xslt	XSL files for form processing

## XBL Components

XBL files are under /WEB-INF/resources/xbl/nul:



XBL Component	Purpose
authoritiesClient	Main Authorities Client XBL component
authoritiesCorporateHeading	XBL component for Corporate Headings
authoritiesLookup	XBL component for autocomplete authority browsing
authoritiesPersonalHeading	XBL component for Personal Headings
css	CSS files
instances	XML templates
logs	
selects	XML select lists
vraSearch	Not in use
vraSet	VRA Set Components
vraWork	Encapsulates thumbnail drag and drop



## Dependencies

The VRA Editor is one component among several that work together to help manage the Digital Image Library. Its integration with these other tools and services also results in various dependencies on their interfaces and protocols to function fully. These dependencies are listed below.

### Fedora

#### Fedora Image Disseminator

Thumbnail images presented in the Editor's search results and Work / Image relationship panels are referenced via the NUL **inu:sdef-image**, Disseminator Service Definition, specifically the **inu:sdef-image/getWithLongSide** method.

#### Fedora VRA Access

**/datastreams/VRA/content**

#### Fedora SVG Access

**/datastreams/DELIV-OPS/content**

#### Image Server Tile Interface

The Flash-based zooming viewer used in the Image panel uses the Aware Tile Server.

#### Authorities Service

The Authorities Service API is documented elsewhere.

#### DIL Hydra REST API

Several functions for searching, saving, copying, and deleting objects in DIL invoke associated Controller Actions in the Hydra-based DIL implementation, currently at **http://cecil.library.northwestern.edu:3000**.

#### Search

**./external\_search/search\_hydra.xml**

#### Save

**./multiresimages/create\_update\_fedora\_object.xml**

#### Clone

**./multiresimages/clone\_work.xml**

#### Delete

**./multiresimages/delete\_fedora\_object.xml**

Display

./multiresimages/