Evergreen 2.1 Documentation

Draft Version

Documentation Interest Group

Evergreen 2.1 Documentation: Draft Version

Documentation Interest Group

This document was updated 2012-05-29.

Table of Contents

List of Tables

Part I. Introduction

The book you're holding in your hands or viewing on a screen is The Book of Evergreen, the official guide to the 2.x version of the Evergreen open source library automation software. This guide was produced by the Evergreen Documentation Interest Group (DIG), consisting of numerous volunteers from many different organizations. The DIG has drawn together, edited, and supplemented pre-existing documentation contributed by libraries and consortia running Evergreen that were kind enough to release their documentation into the creative commons. For a full list of authors and contributing organizations, see Appendix B, *About this Documentation*. Just like the software it describes, this guide is a work in progress, continually revised to meet the needs of its users, so if you find errors or omissions, please let us know, by contacting the DIG facilitators at docs@evergreen-ils.org.

This guide to Evergreen is intended to meet the needs of front-line library staff, catalogers, library administrators, system administrators, and software developers. It is organized into Parts, Chapters, and Sections addressing key aspects of the software, beginning with the topics of broadest interest to the largest groups of users and progressing to some of the more specialized and technical topics of interest to smaller numbers of users.

Copies of this guide can be accessed in PDF and HTML formats from the Documentation section of http://evergreen-ils.org/ and are included in DocBook XML format along with the Evergreen source code, available for download from the same Web site.

Chapter 1. About Evergreen

Evergreen is an open source library automation software designed to meet the needs of the very smallest to the very largest libraries and consortia. Through its staff interface, it facilitates the management, cataloging, and circulation of library materials, and through its online public access interface it helps patrons find those materials.

The Evergreen software is freely licensed under the GNU General Public License, meaning that it is free to download, use, view, modify, and share. It has an active development and user community, as well as several companies offering migration, support, hosting, and development services.

The community's development requirements state that Evergreen must be:

- *Stable*, even under extreme load.
- *Robust*, and capable of handling a high volume of transactions and simultaneous users.
- *Flexible*, to accommodate the varied needs of libraries.
- *Secure*, to protect our patrons' privacy and data.
- *User-friendly*, to facilitate patron and staff use of the system.

Evergreen, which first launched in 2006 now powers over 544 libraries of every type – public, academic, special, school, and even tribal and home libraries – in over a dozen countries worldwide.

Chapter 2. Release Notes

Installation enhancements

- `eg_db_config.pl` now has a `--create-database` option to automatically create the database and add the required contrib modules/extensions, rather than requiring users to carry out a number of error-prone manual steps.

- The OpenILS Perl modules are now installed in the system Perl package directories. In prior versions, they were typically installed in `/openils/lib/perl5`.

- README is now the single source of install instructions

Administration enhancements

- Automatic client updates: It is now possible to build staff clients that, when they connect to the Evergreen server, automatically check to see if the staff client has been updated. If so, the staff client can download just the files that have been updated and replace those files in the existing staff client.

- Printing subsystem: Changes to the staff client should improve the print functionality (both speed and likelihood of success).

- To protect against exploits between a shared staff client session via operator change, add the requirement for the DEBUG_CLIENT permission to invoke debugging functions such as the Javascript Shell or DOM Inspector in the staff client.

Cataloging enhancements

- Unified copy / call number editor: The default holdings maintenance interface now enables cataloguers to edit copies and call numbers on a single screen.

- Call number prefixes and suffixes: While previous releases of Evergreen supported the automatic prepending or appending of prefixes and suffixes to call numbers based on their shelving location, the functionality was limited primarily to printing spine labels. As of 2.1, you also have the option to assign prefixes or suffixes for individual call numbers; these will be displayed in the public catalogue and other interfaces, but prefixes in particular do not affect shelf browsing, which is based entirely on the label.

- Bibliographic parts: This feature adds the ability to designate specific "part" roles for the items attached to a given bibliographic record, such that a user can place holds on a specific kind of item without limiting it to an individual item. For example, libraries may assign parts to each DVD in a season of a popular TV show; or to individual volumes of an encyclopedia. This feature also introduces "part holds".

- Conjoined items: This feature adds the ability to associate a single barcode with multiple bibliographic records, so that the availability of those records is updated based on that copy. For example, libraries may make an electronic reader available for loan that is preloaded with 1000 electronic books; they can add 1000 bibliographic records to the library system and associate them all with a single barcode, so that when the electronic reader is signed out, the availability for all 1000 records changes to "Checked out".

Circulation improvements

- Transfer selected holds to a different title: It is now possible to select multiple holds on a given title and transfer them to a different title. Use case: Your library acquires another copy of "Pride and Prejudice and Zombies", but as it is a new edition it must be cataloged as a new bibliographic record; however, you would like to spread out some of the holds queue from your current edition to the new edition.

- Mark patron as exempt from billing collections: It is now possible to mark a patron as being exempt from being sent to collections.

- Hold-driven recalls: It is now possible to define a loan period threshold (such as 60 days) so that if a hold is placed on an item with a loan period beyond that threshold, the system will automatically shorten the loan period to a specified value; change the fine rules; and send email notification to the person who currently holds the item saying "A hold has been placed on this item, get it back before <new_due_date>".

- Weighting of the individual fields of the in-database circulation and holds rules can now be configured, rather than using hard-coded values. This allows the importance of each field to be adjusted as needed through the circulation matrix weighting, hold matrix weighting, and weighting associations settings.

- The in-database circulation rules support "fall-through" wherein some information can be left out of more specific rules and filled in from less specific rules. For example, disabling fines for staff could be accomplished with a single rule setting only the fine rate and/or max fine rules, without having to duplicate all duration related rules in the system.

- Grace periods are now stored in the database and set via circulation rules, rather than passed into the fine generation code, and can thus differ between different rules. They are also no longer based on a number of fine intervals, but are instead defined by their own specific interval. This can eliminate problems with closed date interpretation during grace periods.

Public interface improvements

- Spell-checking suggestions are now case-insensitive, to avoid generating terms which would yield the same result set.

- Part holds: patrons are guided to select a specific part if they are placing a hold on a bibliographic record that has parts assigned to its copies.

- Located URI visibility: Located URIs (856 fields with a subfield $9 specifying the shortname of the owning library) now show up in a search with a context library of its owning library or below. For example, a Located URI with an owner of SYS1 **will** cause its record to show up in a search with a context OU of BR1 or SYS1, regardless of depth scoping.

- Set a JavaScript cache token when `autogen.sh` is run, so that a subsequent run of `autogen.sh` will automatically cause browsers to fetch refreshed copies of generated files. This improves the usability of the catalog after upgrades or certain configuration changes by preventing browsers from relying on stale cached copies of these files.

Serials enhancements

- Routing lists: it is now possible to define and print routing lists for subscriptions.

- Clone subscriptions: new functionality available in the alternate serials control view to clone subscriptions, thereby speeding initial setup of a lot of subscriptions.
- MFHD/Distribution summary methods: enable the summary method field for generating summary holdings statements from the distribution and/or the record entry (i.e., the MFHD). The four options that are available are:
  - Add to record entry
  - Merge with record entry
  - Use record entry only
  - Do not use record entry
- Advanced receiving: rename the previous serials receiving interface to "advanced receiving". The new "receive" interface will now show items from an entire subscription, but not allow you to receive directly into a specific unit.
- Unit-less receiving: new option for receiving serials into a *no unit*, allowing items to be received but not unitized (i.e., created as a circulating copy record). This allows issues to be received for reading room or non-circulating use.
- Caption/pattern enhancements:
  - The Caption/Pattern Wizard now includes a graphical way to include regularity information (that is, the values typically conveyed in the `85x` subfield `$y`).
  - The Alternate Serials Control interface can now import caption and pattern information from the 85x fields of the MFHD record, allowing the Caption/Pattern Wizard to be bypassed.
- Can now create one-off issuances for unpredicted or unexpected serial issues.
- Improvements to the copy template editor for serials, including the addition of reasonable default values for some fields.
- Various minor interface improvements, including new links between various pages in the serials control interfaces and removing redundant questions from the holding code mini-wizard.
- Bugfixes to the batch receive interface.

Staff client improvements

- The client supports new hotkey sets that are selectable per workstation. In addition, there is now a toggle for temporarily disabling hotkeys available on the toolbar and in the admin menu.
- The client has a second toolbar for cataloging functions as well as options controlling the presence and size of icons and presence and position of labels. To support the preferences of individual staff, each workstation can be configured differently. See the **Admin # Workstation Administration # Toolbars** sub-menu for options.
- Patron registration now supports "Suggested" fields that are configurable per library. In addition, library settings allow required fields to be adjusted, and it is possible to specify regular expressions to validate input for many fields in the patron registration interface. Open **Admin # Local Administration # Library Settings Editor** and filter on `GUI`.

New configuration and administration settings

New `opensrf.xml` settings

(The path to these settings is relative to `/opensrf/default/apps/`).

- `open-ils.search/app_settings/default_CD_modifiers`: parameters to the cover density ranking function used to calculate relevance during bibliographic searches.
- `open-ils.trigger/app_settings/parallel/collect`: number of parallel processes that `action_trigger_runner.pl` should use when collecting events. Setting this can decrease the time it takes for Action/Trigger processing to run on large Evergreen databases.
- `open-ils.trigger/app_settings/parallel/react`: number of parallel processes that `action_trigger_runner.pl` should use when processing event reactors. Setting this can decrease the time it takes for Action/Trigger processing to run on large Evergreen databases.
- `open-ils.resolver/app_settings/cache_timeout`: how long to cache results from your OpenURL resolver.
- `open-ils.resolver/app_settings/default_url_base`: set to the base URL of your OpenURL resolver.

In addition, the `default_preferred_language` and `default_preferred_language_weight` settings have been moved from the `open-ils.storage` simplesect to the `open-ils.search` simplesect of `opensrf.xml`.

New administration pages

The following new administration pages can all be accessed from the `Admin | Server Administration` menu item in the staff client:

- Call Number Prefixes: Populates a drop down menu in the new Unified Copy / Call Number Editor
- Call Number Suffixes: Populates a drop down menu in the new Unified Copy / Call Number Editor
- Circulation Matchpoint Weights: Related to the in-database Circulation and Hold Rule configuration changes
- Hold Matchpoint Weights: Related to the in-database Circulation and Hold Rule configuration changes
- MARC Coded Value Maps: Sets labels for fixed fields and extends the set of values for display
- MARC Record Attributes: Sets labels for fixed fields and extends the set of values for display
- Weights Associations: Related to the in-database Circulation and Hold Rule configuration changes

New library settings

Label	Description
Cataloging: Default copy status (fast add)	Default status when a copy is created using the "Fast Add" interface.

Label	Description
Cataloging: Default copy status (normal)	Default status when a copy is created using the normal volume/copy creator interface.
GUI: Default Hotkeyset	Default Hotkeyset for clients (filename without the .keyset). Examples: Default, Minimal, and None
GUI: Default showing suggested patron registration fields	Instead of All fields, show just suggested fields in patron registration by default.
GUI: Example for day_phone field on patron registration	The Example for validation on the day_phone field in patron registration.
GUI: Example for email field on patron registration	The Example for validation on the email field in patron registration.
GUI: Example for evening_phone field on patron registration	The Example for validation on the evening_phone field in patron registration.
GUI: Example for other_phone field on patron registration	The Example for validation on the other_phone field in patron registration.
GUI: Example for phone fields on patron registration	The Example for validation on phone fields in patron registration. Applies to all phone fields without their own setting.
GUI: Example for post_code field on patron registration	The Example for validation on the post_code field in patron registration.
GUI: Horizontal layout for Volume/Copy Creator/Editor.	The main entry point for this interface is in Holdings Maintenance, Actions for Selected Rows, Edit Item Attributes / Call Numbers / Replace Barcodes. This setting changes the top and bottom panes for that interface into left and right panes.
GUI: Regex for day_phone field on patron registration	The Regular Expression for validation on the day_phone field in patron registration.
GUI: Regex for email field on patron registration	The Regular Expression for validation on the email field in patron registration.
GUI: Regex for evening_phone field on patron registration	The Regular Expression for validation on the evening_phone field in patron registration.
GUI: Regex for other_phone field on patron registration	The Regular Expression for validation on the other_phone field in patron registration.
GUI: Regex for phone fields on patron registration	The Regular Expression for validation on phone fields in patron registration. Applies to all phone fields without their own setting.
GUI: Regex for post_code field on patron registration	The Regular Expression for validation on the post_code field in patron registration.
GUI: Require county field on patron registration	The county field will be required on the patron registration screen.
GUI: Require day_phone field on patron registration	The day_phone field will be required on the patron registration screen.
GUI: Require dob field on patron registration	The dob field will be required on the patron registration screen.
GUI: Require email field on patron registration	The email field will be required on the patron registration screen.

Label	Description
GUI: Require evening_phone field on patron registration	The evening_phone field will be required on the patron registration screen.
GUI: Require other_phone field on patron registration	The other_phone field will be required on the patron registration screen.
GUI: Show active field on patron registration	The active field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show alert_message field on patron registration	The alert_message field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show alias field on patron registration	The alias field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show barred field on patron registration	The barred field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show calendar widget for dob field on patron registration	If set the calendar widget will appear when editing the dob field on the patron registration form.
GUI: Show claims_never_checked_out_count field on patron registration	The claims_never_checked_out_count field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show claims_returned_count field on patron registration	The claims_returned_count field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show day_phone field on patron registration	The day_phone field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show dob field on patron registration	The dob field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show email field on patron registration	The email field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show evening_phone field on patron registration	The evening_phone field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show ident_value field on patron registration	The ident_value field will be shown on the patron registration screen. Showing a field makes it appear with required fields

Label	Description
	even when not required. If the field is required this setting is ignored.
GUI: Show ident_value2 field on patron registration	The ident_value2 field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show juvenile field on patron registration	The juvenile field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show master_account field on patron registration	The master_account field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show other_phone field on patron registration	The other_phone field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show second_given_name field on patron registration	The second_given_name field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Show suffix field on patron registration	The suffix field will be shown on the patron registration screen. Showing a field makes it appear with required fields even when not required. If the field is required this setting is ignored.
GUI: Suggest active field on patron registration	The active field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest alert_message field on patron registration	The alert_message field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest alias field on patron registration	The alias field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest barred field on patron registration	The barred field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest claims_never_checked_out_count field on patron registration	The claims_never_checked_out_count field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest claims_returned_count field on patron registration	The claims_returned_count field will be suggested on the patron registration screen. Suggesting a field makes it appear

Label	Description
	when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest day_phone field on patron registration	The day_phone field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest dob field on patron registration	The dob field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest email field on patron registration	The email field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest evening_phone field on patron registration	The evening_phone field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest ident_value field on patron registration	The ident_value field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest ident_value2 field on patron registration	The ident_value2 field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest juvenile field on patron registration	The juvenile field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest master_account field on patron registration	The master_account field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest other_phone field on patron registration	The other_phone field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest second_given_name field on patron registration	The second_given_name field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Suggest suffix field on patron registration	The suffix field will be suggested on the patron registration screen. Suggesting a field makes it appear when suggested fields are shown. If the field is shown or required this setting is ignored.
GUI: Unified Volume/Item Creator/Editor	If true combines the Volume/Copy Creator and Item Attribute Editor in some instances.

Label	Description
Printing: Custom Javascript File	Full URL path to a Javascript File to be loaded when printing. Should implement a print_custom function for DOM manipulation. Can change the value of the do_print variable to false to cancel printing.
Recalls: An array of fine amount, fine interval, and maximum fine.	Recalls: An array of fine amount, fine interval, and maximum fine. For example, to specify a new fine rule of $5.00 per day, with a maximum fine of $50.00, use: [5.00,"1 day",50.00]
Recalls: Circulation duration that triggers a recall.	Recalls: A hold placed on an item with a circulation duration longer than this will trigger a recall. For example, "14 days" or "3 weeks".
Recalls: Truncated loan period.	Recalls: When a recall is triggered, this defines the adjusted loan period for the item. For example, "4 days" or "1 week".

New user permissions

Code	Description
ADMIN_CODED_VALUE	Create/Update/Delete SVF Record Attribute Coded Value Map
ADMIN_SERIAL_ITEM	Create/Retrieve/Update/Delete Serial Item
ADMIN_SVF	Create/Update/Delete SVF Record Attribute Defintion
CREATE_BIB_PTYPE	Create Bibliographic Record Peer Type
CREATE_MONOGRAPH_PART	Create monograph part definition.
CREATE_PURCHASE_REQUEST	Create User Purchase Request
CREATE_VOLUME_PREFIX	Create prefix label definition.
CREATE_VOLUME_SUFFIX	Create suffix label definition.
DEBUG_CLIENT	Allows a user to use debug functions in the staff client
DELETE_BIB_PTYPE	Delete Bibliographic Record Peer Type
DELETE_MONOGRAPH_PART	Delete monograph part definition.
DELETE_VOLUME_PREFIX	Delete prefix label definition.
DELETE_VOLUME_SUFFIX	Delete suffix label definition.
MAP_MONOGRAPH_PART	Create/Update/Delete Copy Monograph Part Map
MARK_ITEM_MISSING_PIECES	Allows the Mark Item Missing Pieces action.
UPDATE_BIB_PTYPE	Update Bibliographic Record Peer Type
UPDATE_HOLD_REQUEST_TIME	Allows editing of a hold’s request time, and/or its Cut-in-line/Top-of-queue flag.
UPDATE_MONOGRAPH_PART	Update monograph part definition.
UPDATE_PATRON_COLLECTIONS_EXEMPT	Allows a user to indicate that a patron is exempt from collections processing
UPDATE_PICKLIST	Allows update/re-use of an acquisitions pick/selection list.
UPDATE_VOLUME_PREFIX	Update prefix label definition.
UPDATE_VOLUME_SUFFIX	Update suffix label definition.

Code	Description
UPDATE_WORKSTATION	Allows update of a workstation during workstation registration override.
VIEW_USER_SETTING_TYPE	Allows viewing of configurable user setting types.

Part II. Public Access Catalog

This part of the documentation explains how to use the Evergreen public OPAC. It covers the basic catalog and more advanced search topics. It also describes the “My Account” tools users have to find information and manage their personal library accounts through the OPAC. This section could be used by staff and patrons but would be more useful for staff as a generic reference when developing custom guides and tutorials for their users.

Part III. Core Staff Tasks

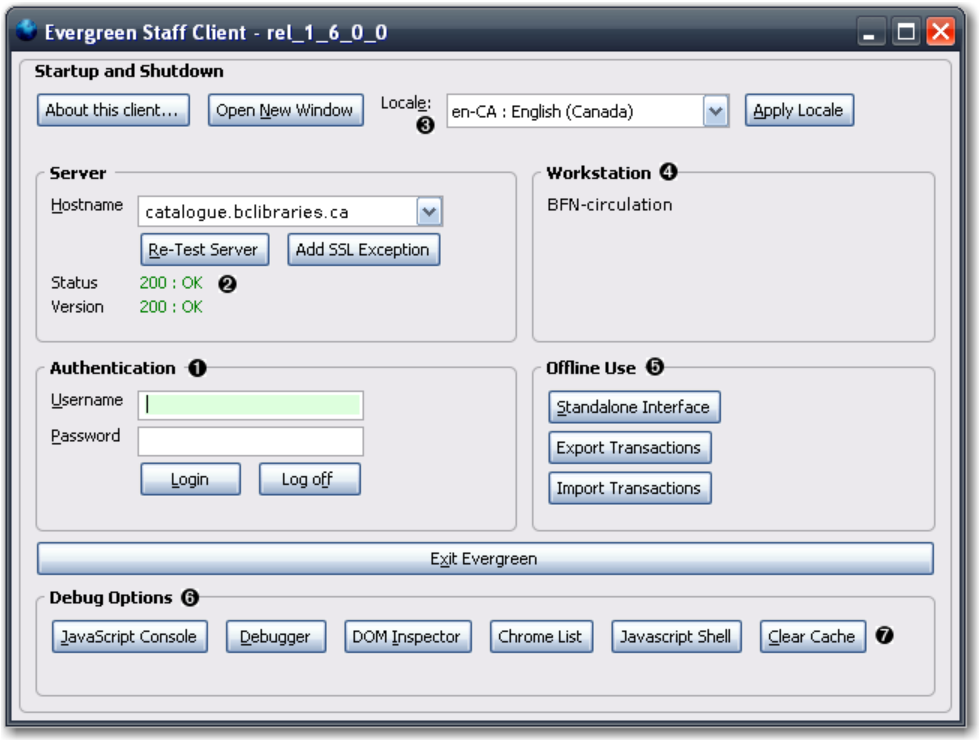
This part of the documentation covers a broad range of the common tasks carried out by your library and includes tasks performed by circulation staff and catalogers among others. Some of these procedures should only be performed by Local System Administrators, but most of these sections will give all staff a better understanding of the Evergreen system and its features.

Chapter 3. Using the Staff Client

Logging in to Evergreen

To log in you must first install the Evergreen Staff Client, available for download from the Evergreen site at http://downloads.open-ils.org/.

Each staff member can have their own username and password, or generic logins can be used.



❶ Enter Username and Password for your staff account, then click Login. Under normal circumstances this is all that is required to login.

❷ If the staff client can connect to Evergreen both Status and Version display a green *200:OK* message. If not, ensure the hostname is correctly entered and click Re-Test Server. If the error message persists make sure your are connected to the internet.

❸ Locale sets the language preferences for the staff client.

❹ Workstation identifies your physical computer location. Workstation registration is done by a Local System Administrator when staff clients are first installed.

❺ If your connection to Evergreen is lost during open hours, click Standalone Interface to continue with check out and patron registration functions until the connection is restored.

❻ Debug Options are for advanced troubleshooting and can be ignored in normal use.

❼ Click Clear Cache to remove the staff client's locally cached files. This may be required to see recent changes to administrative settings.

Navigation

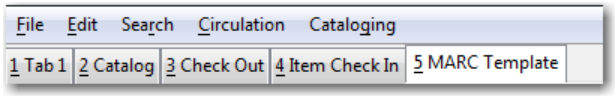
Tabs

Evergreen uses tabs to display functions. Tabs allow all software functionality to be open in one window. You can have up to 9 tabs open at once and you can have more than one tab of a single function open at the same time. You simply move through the tabs to perform your work.

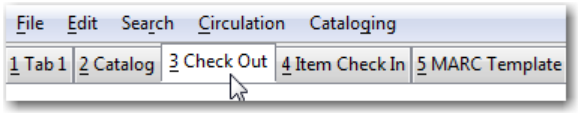
Keyboard shortcuts for working with tabs:

- **Ctrl**+**T** new tab
- **Ctrl**+**W** close tab
- **Ctrl**+**Shift**+**W** close all tabs
- **Ctrl**+**Tab** tabs forward through open tabs
- **Ctrl**+**Shift**+**Tab** tabs backward through open tabs

In the example below, the MARC Template tab is active. Click on any open tab to bring that screen to the front. You can also use **Ctrl**+**Tab** to move to the required tab



Now the Check Out tab is the active screen.



Once you are in the selected tab, you can use the drop down menus or keyboard shortcuts to perform required functions. Menu functions and corresponding keyboard shortcuts will be demonstrated throughout this manual.

Keyboard Shortcuts

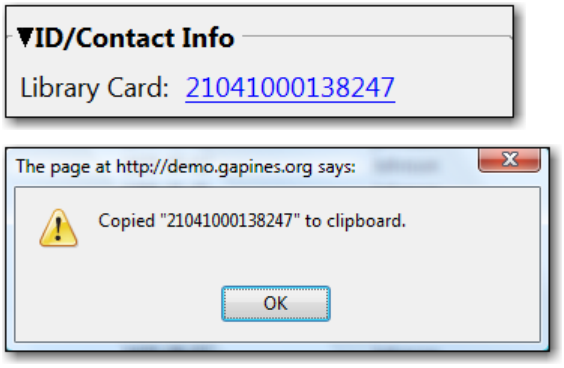
Most menu items have keyboard shorcuts that can greatly increase efficiency. Below is a selected list of commonly used shortcut keys:

Key	Function
F1	Checkout, or retrieve patron record by barcode
F2	Checkin
F3	Catalogue search
F4	Patron search
F5	Retrieve copy by barcode
F6	Record in house use
F8	Retreive last patron
F9	Re-print the last receipt
Shift+F1	Register new patron
Shift+F2	Capture holds
Shift+F3	Retrieve record by TCN
Shift+F8	Retreive last patron
Ctrl+T	Open new tab
Ctrl+W	Close current tab
Ctrl+Tab	Move forward through tabs
Ctrl+Shift+Tab	Move back through tabs
Ctrl+C	Copy
Ctrl+V	Paste

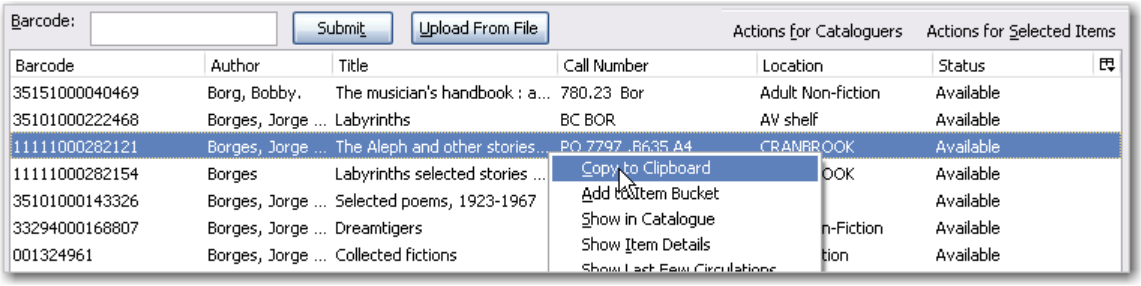
Copy/Paste

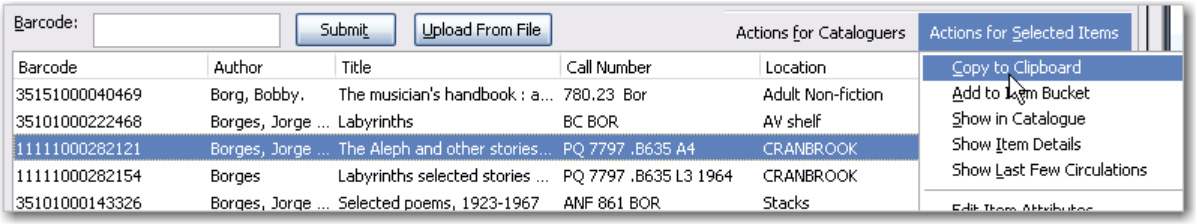
There are several methods of copying and pasting text in Evergreen, depending on where you are in the staff client and the type of information you are copying

1. **Underlined blue text.** Clicking on any of the blue links in the Evergreen client copies the data to the computer clipboard (left and right click work the same way for these links). To paste into another location, use **Ctrl+V**.

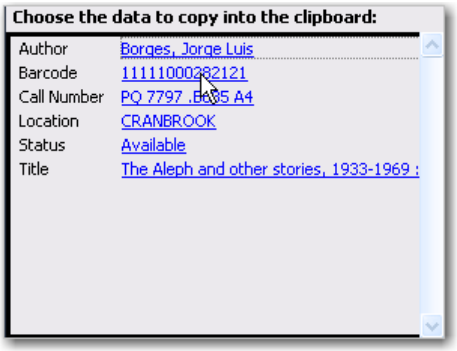


2. **Text displayed in tables.** To copy information from a staff client table, first select the desired row then right-click and choose Copy to Clipboard; alternatively select Actions for Selected Items → Copy to Clipboard.

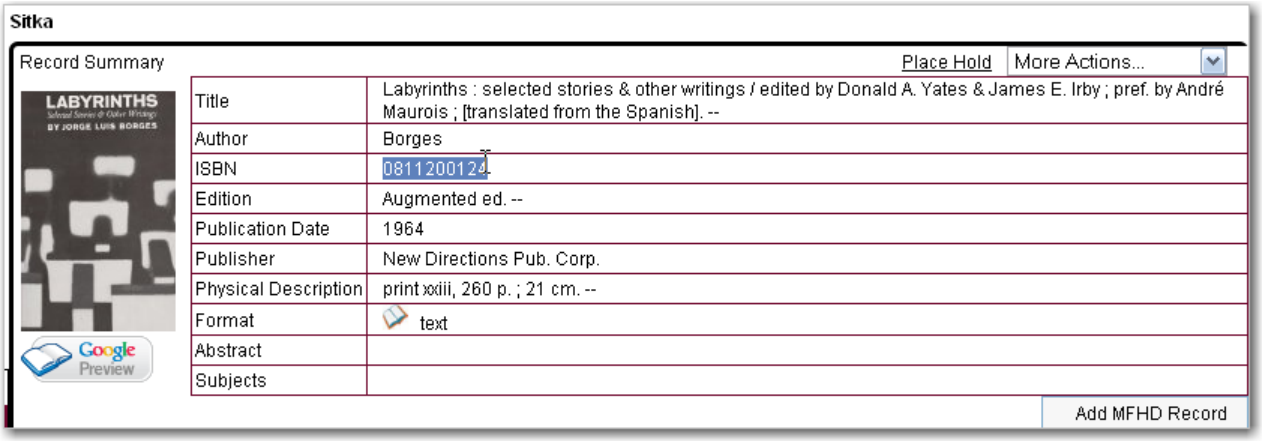




Next click the desired information in the popup to copy it to the clipboard



3. **Text from catalogue search results.** There is no right-click menu for copying data from staff client search results. To copy the ISBN in the example below, highlight it and click **Ctrl**+**C**. To paste into another location use **Ctrl**+**V**.

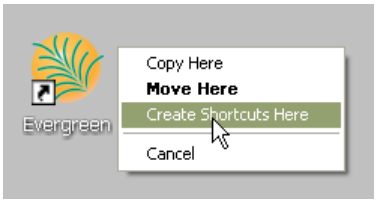


Preset Tabs in Evergreen Client

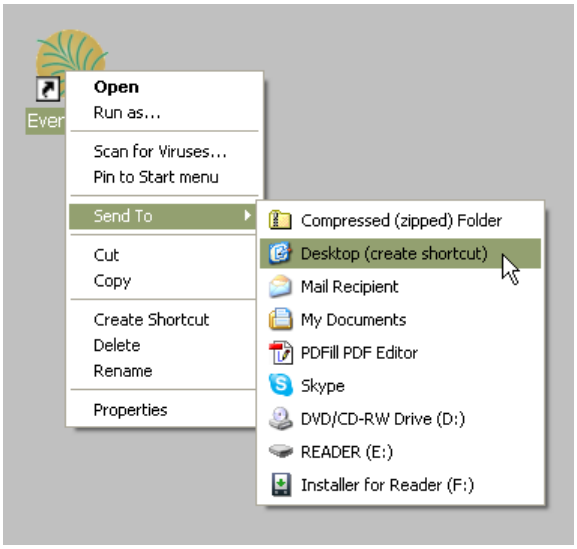
To have preset tabs waiting when Evergreen opens you will need to modify the **Evergreen shortcut** on your desktop.

1) First, you need to copy your shortcut. There are a couple of ways to do this.

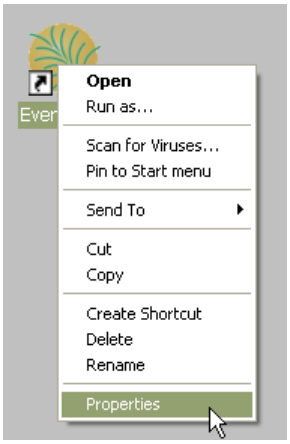
A) Right-mouse click and drag icon; upon release select Create Shortcut Here.



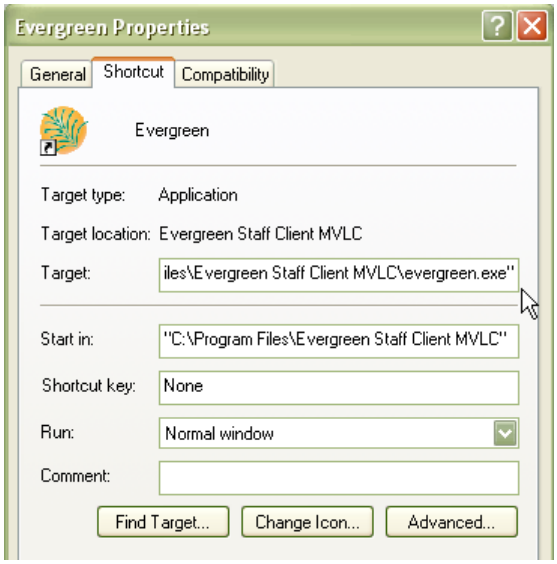
B) Right-mouse click icon, select Send to, and select Desktop (create shortcut).



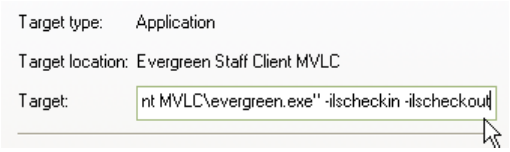
2) Right-mouse click the new shortcut and select Properties.



3) Listed in the **Target** box you will see something like the following path surrounded by quotation marks: **C:\Program Files\Evergreen Staff Client MVLC\evergreen.exe**



4) Place your cursor after the ending quotation mark. Enter a space after the last quotation mark and then enter the tab code from the list below. Add multiple tabs with a space separating them.



For Example, to have Checkout, Checkin, Catalog Search, and a blank tab preset: **C:\Program Files\Evergreen Staff Client MVLC\evergreen.exe" -ilscheckout -ilscheckin -ilsurl XUL_OPAC_WRAPPER -ilstab**

5) The following options are available:

- -ILScheckin : Opens the Check In interface
- -ILScheckout : Opens the Check Out interface
- -ILSurl <url/constant> : Opens the specified page
- -ILSnew : Opens a new “menu” window
- -ILStab : Opens a new (default) tab
- -ILSnew_default : Opens a new “menu” window with a guaranteed default tab
- -ILSoffline/-ILSstandalone : Opens the standalone interface
- -ILSlogin : Opens the login page

Examples

Useful Tab Codes:

- -ilsurl XUL_PATRON_DISPLAY : Opens a Patron Search tab
- -ilsurl XUL_HOLD_PULL_LIST : Opens a Pull List tab

- -ilsurl XUL_HOLDS_BROWSER : Opens a Browse Holds Shelf tab
- -ilsurl XUL_OPAC_WRAPPER : Opens an Advanced Catalog search tab
- -ilsurl XUL_COPY_STATUS : Opens an Item status by barcode tab
- -ilsurl XUL_RECORD_BUCKETS : Opens a Manage Record Buckets tab
- -ilsurl XUL_COPY_BUCKETS : Opens a Manage Copy Buckets tab
- -ilsurl XUL_MARC_NEW : Opens a Create new MARC record tab
- -ilsurl XUL_Z3950_IMPORT : Opens an Import record from Z39.50 tab
- To open two windows, one with checkin and checkout, one with Marc and Z39.50, use:

  -ilscheckin -ilscheckout -ilsnew -ilsurl XUL_MARC_NEW -ilsurl XUL_Z3950_IMPORT

6) You may want to rename your shortcut to reflect its purpose. For example, you could have one icon set to open circulation-related tabs and one icon to open cataloging-related tabs. Right-mouse click and select Rename to do this.

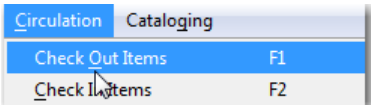
Chapter 4. Circulation

Circulating Items

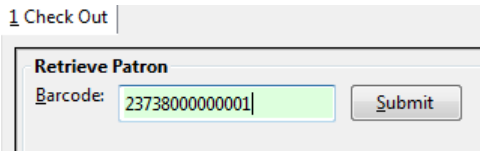
Check Out (F1)

Regular Items

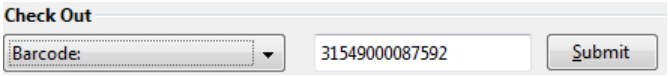
1) To check out an item press **F1**, click **Check Out** on the Circulation toolbar, or select **Circulation** # **Check Out Items**.



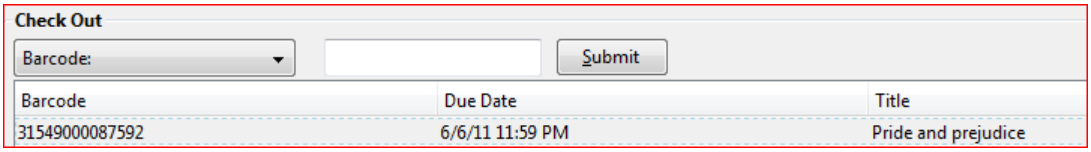
2) Scan or enter patron's barcode and click **Submit** if entering barcode manually. If scanning, number is submitted automatically.



3) Scan or enter item barcode manually, clicking **Submit** if manual.



4) Due date is now displayed.



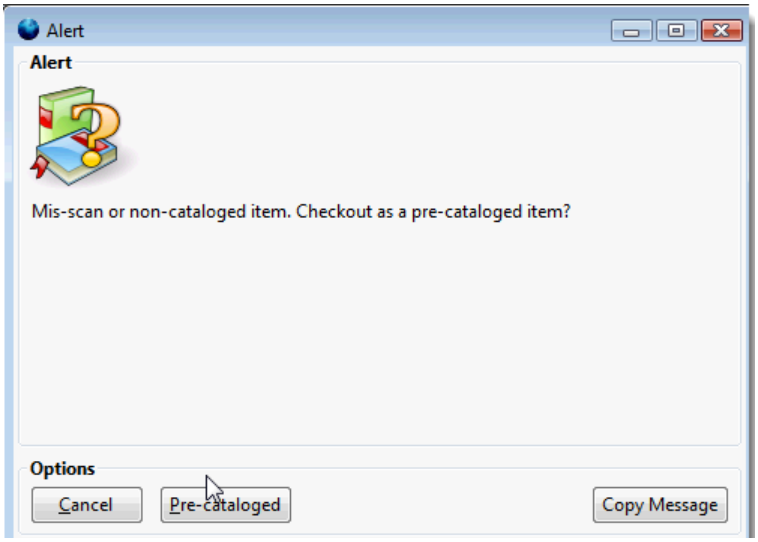
5) When all items are scanned, hit the **F1** key or click the **Check Out** button on the Circulation toolbar to generate slip receipt or to exit patron record if not printing slip receipts.

Pre-cataloged Items

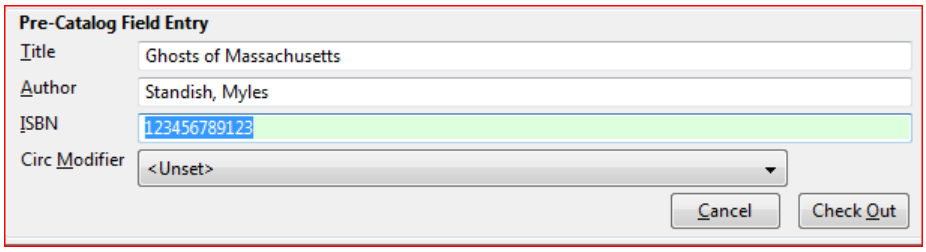
1) Go to patron's **Check Out** screen by clicking **Circulation** # **Check Out Items**.

2) Scan the item barcode.

3) At prompt, click **Pre-Cataloged**.



4) Enter required information and click **Check Out**.

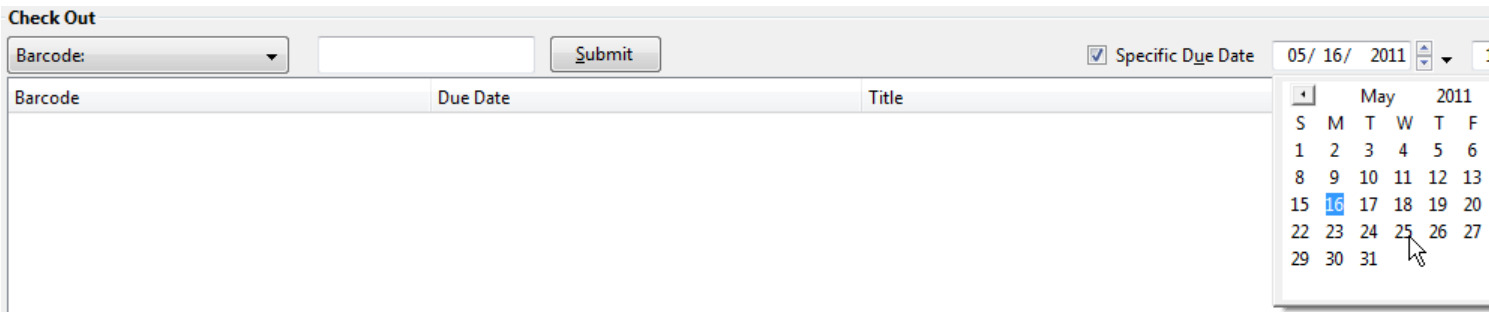


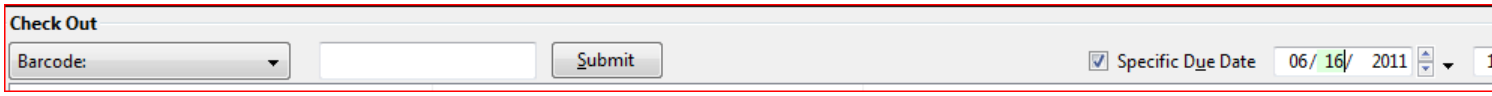
On check-in, Evergreen will prompt staff to re-route the item to cataloging.

Due Dates

Circulation periods are pre-set. When items are checked out, due dates are automatically calculated and inserted into circulation records if the **Specific Due Date** checkbox is not selected on the Check Out screen. The **Specific Due Date** checkbox allows you to set a different due date to override the pre-set loan period.

Before you scan the item, select the **Specific Due Date** checkbox. Use the calendar widget to select a date. Or click in day, month or year, then use the up or down arrows to make the change or simply delete the data, then enter again. Time is used for hourly loan only. This date applies to all items until you change the date, de-select the **Specific Due Date** checkbox, or quit the patron record.

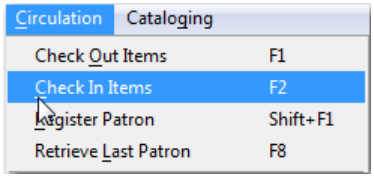




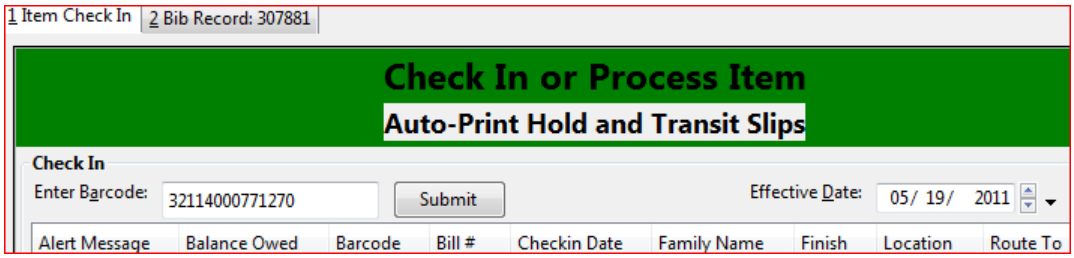
Check In (F2)

Regular check in

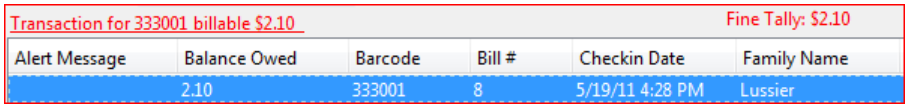
1) To check in an item, select **Circulation # Check In Items**, click **Check In** on the Circulation toolbar, or press **F2**.



2) Scan item barcode or enter manually and click **Submit**.



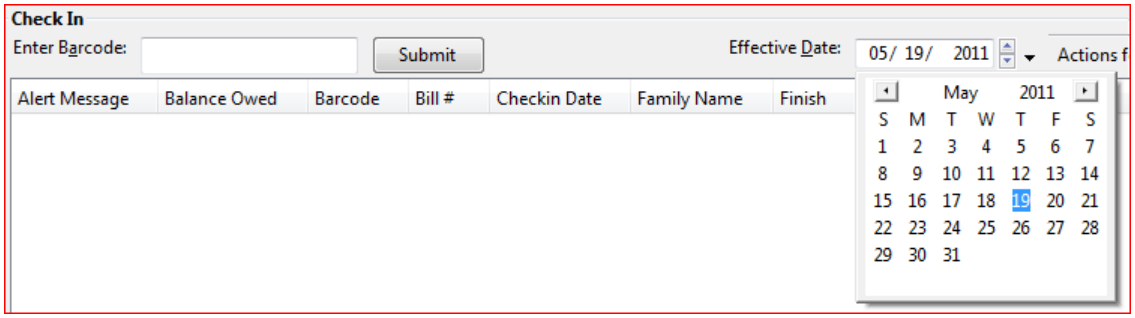
3) If there is an overdue fine associated with the checkin, an alert will appear at the top of the screen with a fine tally for the current checkin session. To immediately handle fine payment, click the alert to jump to the patron's bill record.



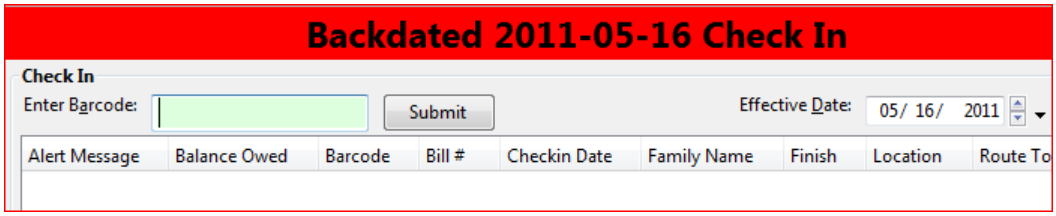
Backdated check in

This is useful for clearing a book drop.

1) To change effective check-in date, select **Circulation** # **Check In Items**, or press **F2**. Use the calendar widget to choose the effective date.



2) The top green bar changes to red. The new effective date is now displayed in the header.

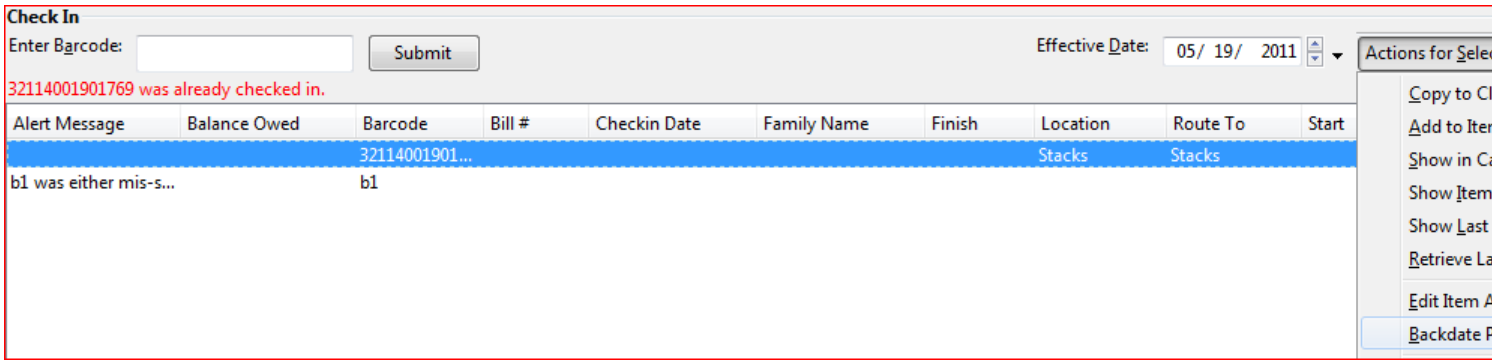


3) Move the cursor to the **Barcode** field. Scan the items. When finishing backdated check-in, change the **Effective Date** back to today's date.

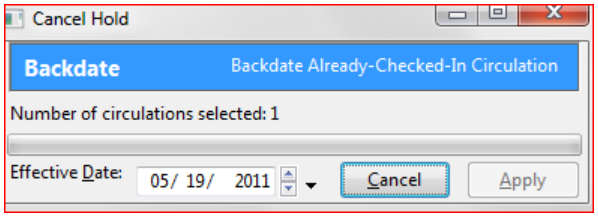
Backdate Post-Checkin

After an item has been checked in, you may use the Backdate Post-Checkin function to backdate the check-in date.

1) Select the item on the Check In screen, click Actions for Selected Items # Backdate Post-Checkin.



2) Use the calendar widget to select an effective check-in date. Click Apply. Overdue fines, if any, will be adjusted according to the new effective check-in date.



Checkin Modifiers

At the right bottom corner there is a **Checkin Modifiers** pop-up list. The options are:

- Ignore Pre-cat Items: no prompt when checking in a pre-cat item. Item will be routed to Cataloging with Cataloging status.
- Suppress Holds and Transit: item will not be used to fill holds or sent in transit. Item has Reshelving status.
- Amnesty Mode/Forgive Fines: overdue fines will be voided if already created or not be inserted if not yet created (e.g. hourly loans).
- Auto-Print Hold and Transit Slips: slips will be automatically printed without prompt for confirmation.

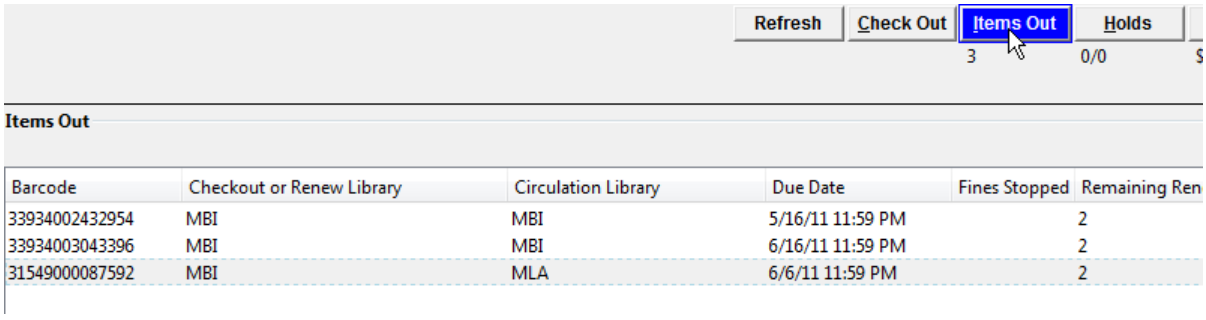
These options may be selected simultaneously. The selected option is displayed in the header area.

Renewal and Editing the Item's Due Date

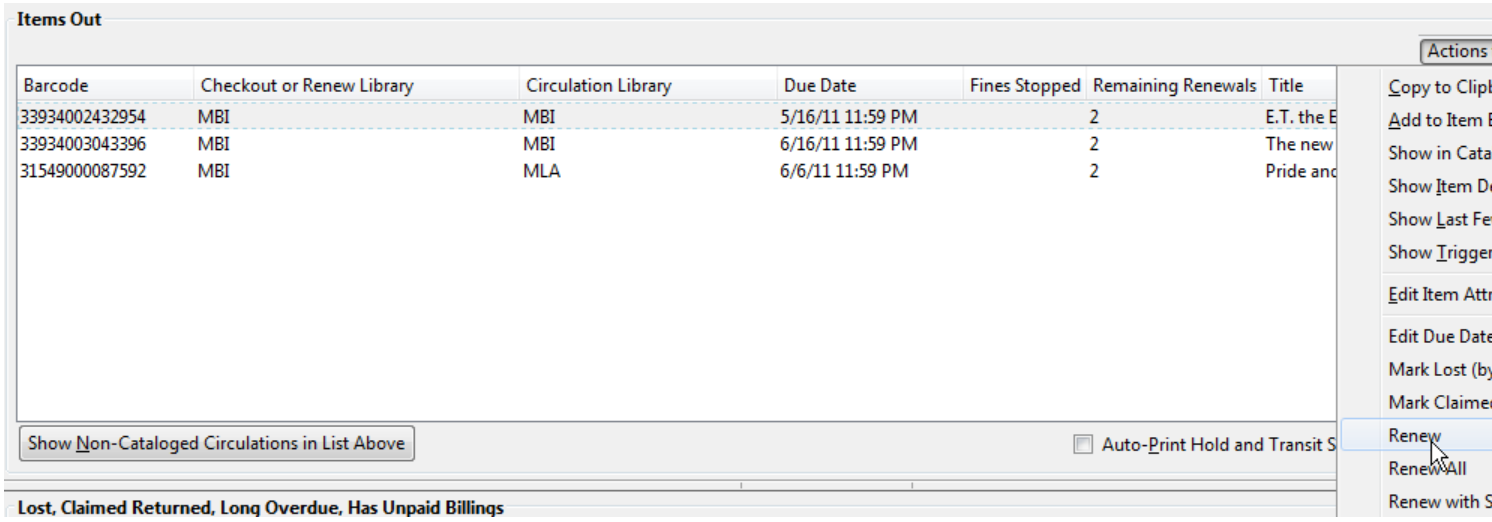
Checked-out items can be renewed if your library's policy allows it. The new due date is calculated from the renewal date. Existing loans can also be extended to a specific date by editing the due date or renewing with a specific due date.

Renewing via a Patron's Account

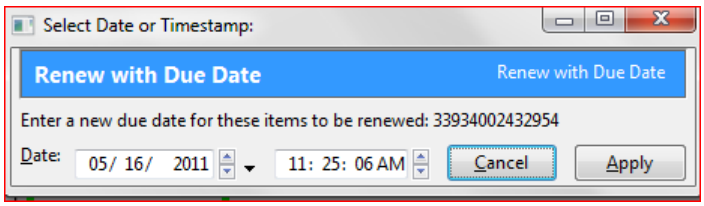
1) Retrieve the patron record and go to the **Items Out** screen.

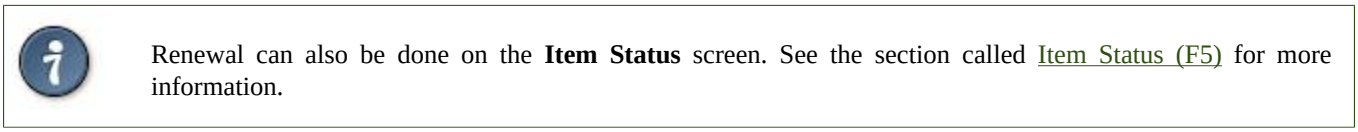


2) Select the item you want to renew. **Click on Actions for Selected Items # Renew**. If you want to renew all items in the account, click **Renew All** instead.



3) If you want to specify the due date, click **Renew with Specific Due Date**. You will be prompted to select a due date. Once done, click **Apply**.



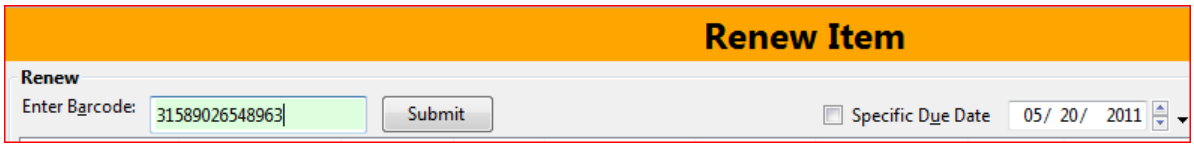


Renewal can also be done on the **Item Status** screen. See the section called Item Status (F5) for more information.

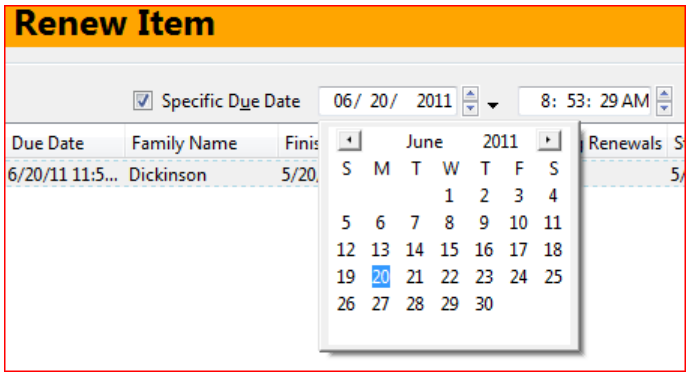
Renewing by Item Barcode

1) To renew items by barcode, select **Circulation** # **Renew Items** or press **CTRL-F2**.

2) Scan or manually entire the item barcode.



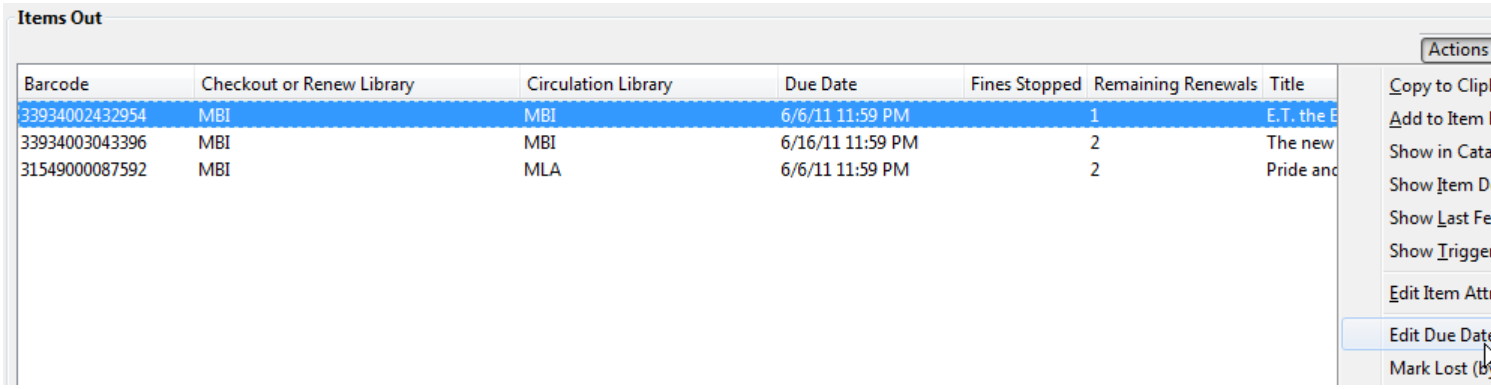
3) If you want to specify the due date, click **Specific Due Date** and select a new due date from the calendar.



Editing Due Date

1) Retrieve the patron record and go to the **Items Out** screen.

2) Select the item you want to renew. Click on **Actions for Selected Items # Edit Due Date**.



3) Select a new due date in the pop-up window, then click **Apply**.

You can select multiple items by pressing down the CTRL key on your keyboard and clicking each items you want to edit.

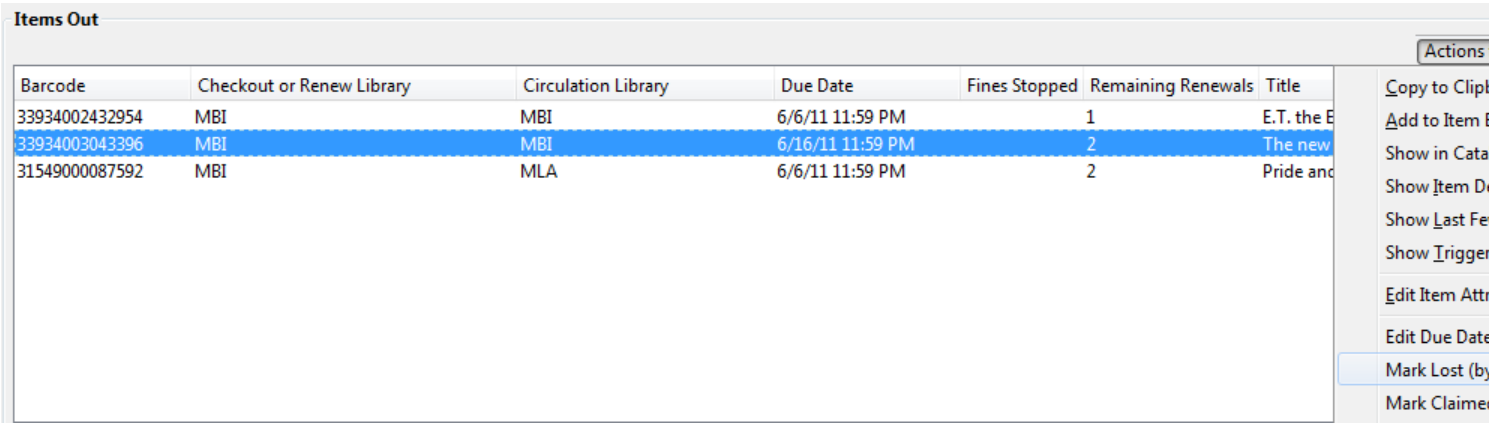
Editing a due date is not included in the renewal count.

Marking Items Lost and Claimed Returned

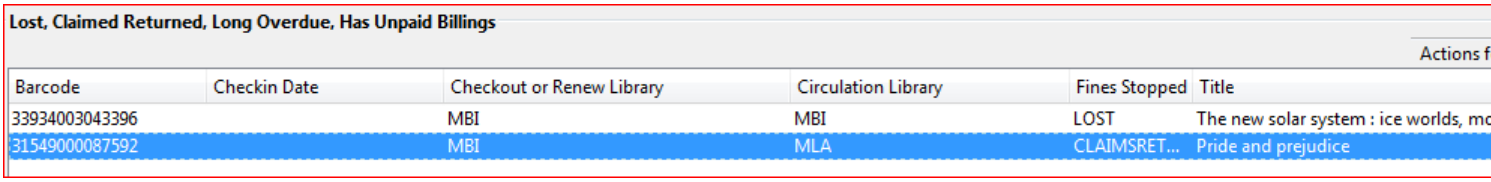
Lost Items

1) To mark items Lost, retrieve patron record and click **Items Out**.

2) Select the item. Click on **Actions for Selected Items # Mark Lost (by Patron)**.



3) The lost item now displays in the **Lost/Claimed Returned/Long Overdue, Has Unpaid Billings** section of the patron record.



4) The lost item also adds to the count of **Lost** items in the patron summary on the left (or top) of the screen.

Lost Item Billing

- Marking an item Lost will automatically bill the patron the replacement cost of the item as recorded in the price field in the item record, and a processing fee as determined by your local policy. If the lost item has overdue charges, the overdue charges may be voided or retained based on local policy.
- A lost-then-returned item will disappear from the Items Out screen only when all bills linked to this particular circulation have been resolved. Bills may include replacement charges, processing fees, and manual charges added to the existing bills.
- The replacement fee and processing fee for lost-then-returned items may be voided if set by local policy. Overdue fines may be reinstated on lost-then-returned items if set by local policy.

Refunds for Lost Items

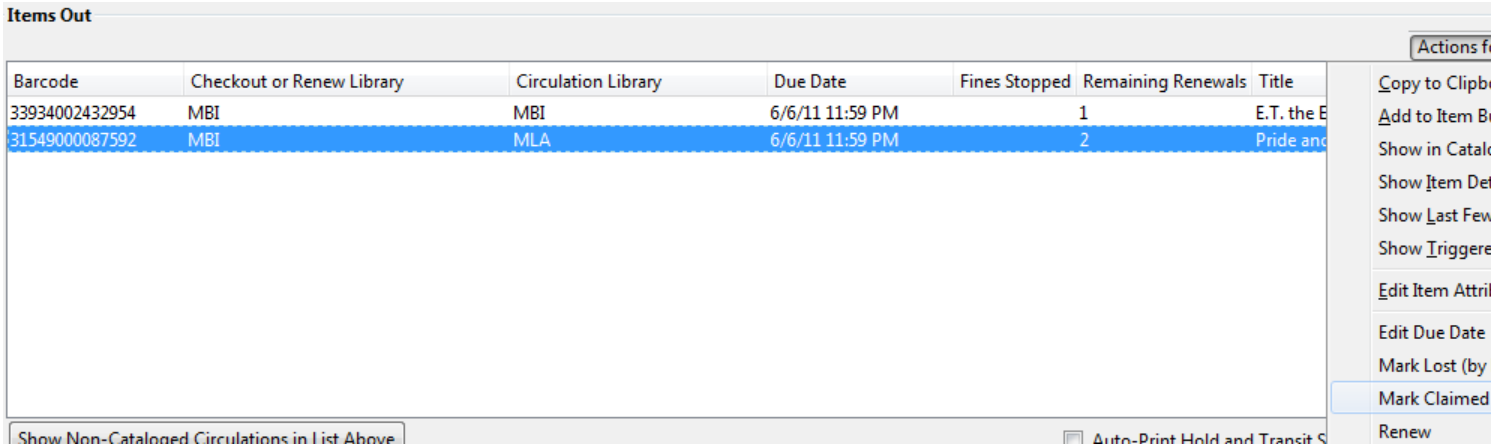
If an item is returned after a lost bill has been paid and the library's policy is to void the replacement fee for lost-then-returned items, there will be a negative balance in the bill. A refund needs to be made to close the bill and the circulation record. Once the outstanding amount has been refunded, the bill and circulation record will be closed and the item will disappear from the Items Out screen.

If you need to balance a bill with a negative amount and close the linked lost circulation record without making a refund (removing the item from the **Lost, Claimed Returned, Long Overdue, Has Unpaid Bills** panel on the **Items Out** screen), you need to add two dummy bills to the existing bills. The first one can be of any amount (e.g. $0.01), while the second should be of the absolute value of the negative amount. Then you need to void the first dummy bill. The reason for using a dummy bill is that Evergreen will check and close the circulation record only when payment is applied or bills are voided.

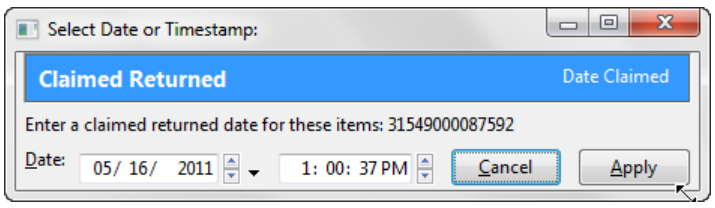
Claimed Returned Items

1) To mark an item Claimed Returned, retrieve the patron record and go to the **Items Out** screen.

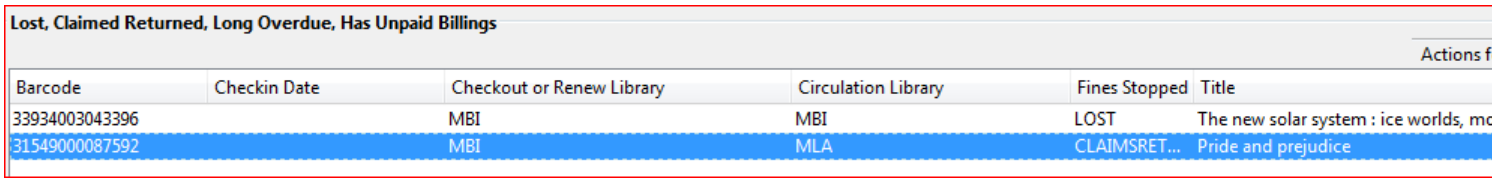
2) Select the item, then select **Actions for Selected Items** # **Mark Claimed Returned** from the dropdown menu.



3) Select a date and click **Apply**.



4) The Claimed Returned item now displays in the **Lost/Claimed Returned/Long Overdue, Has Unpaid Billings** section of the patron record.



5) The Claimed Returned item adds to the count of Check Outs that are Claimed Returned in the patron summary on the left (or top) of the screen. It also adds to the total **Claims-returned Count** (including those that are current Check Outs and those that have since been returned) that is displayed when editing the patron's record.

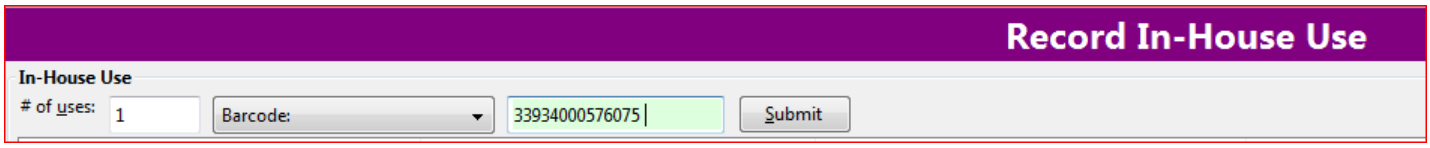
More on Claimed Returned Items

- The date entered for a Claimed Returned item establishes the fine. If the date given has passed, bills will be adjusted accordingly.
- When a Claimed Returned item is returned, if there is an outstanding bill associated with it, the item will not disappear from the **Items Out** screen. It will disappear when the outstanding bills are resolved.
- When an item is marked Claimed Returned, the value in **Claims-returned Count** field in the patron record is automatically increased. Staff can manually adjust this count by editing the patron record.

In-house Use (F6)

1) To record in-house use, select **Circulation** # **Record-In House Use**, click **Check Out** # **Record In-House Use** on the circulation toolbar , or press **F6**.

2) To record in-house use for cataloged items, enter number of uses, scan barcode or type barcode and click **Submit**.



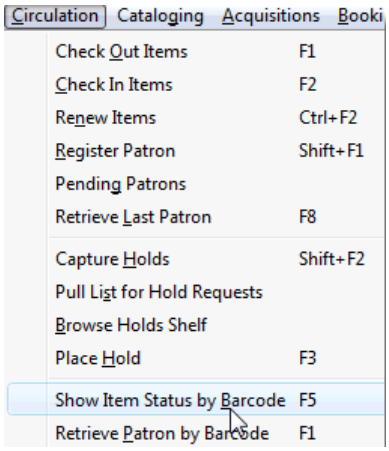
The statistics of in-house use are separated from circulation statistics. The in-house use count of cataloged items is not included in the items' total use count.

Item Status (F5)

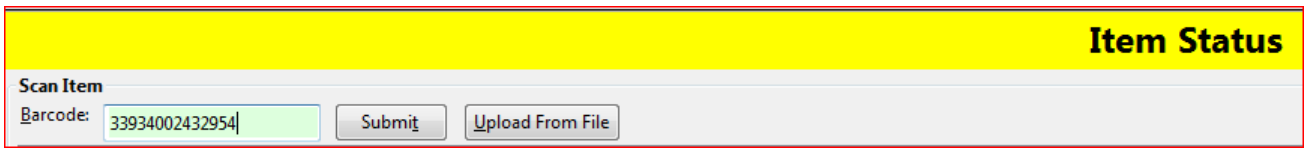
The Item Status screen is very useful. Many actions can be taken by either circulation staff or catalogers on this screen. Here we will cover some circulation-related functions, namely checking item status, viewing past circulations, inserting item alert messages, marking items missing or damaged, etc.

Checking item status

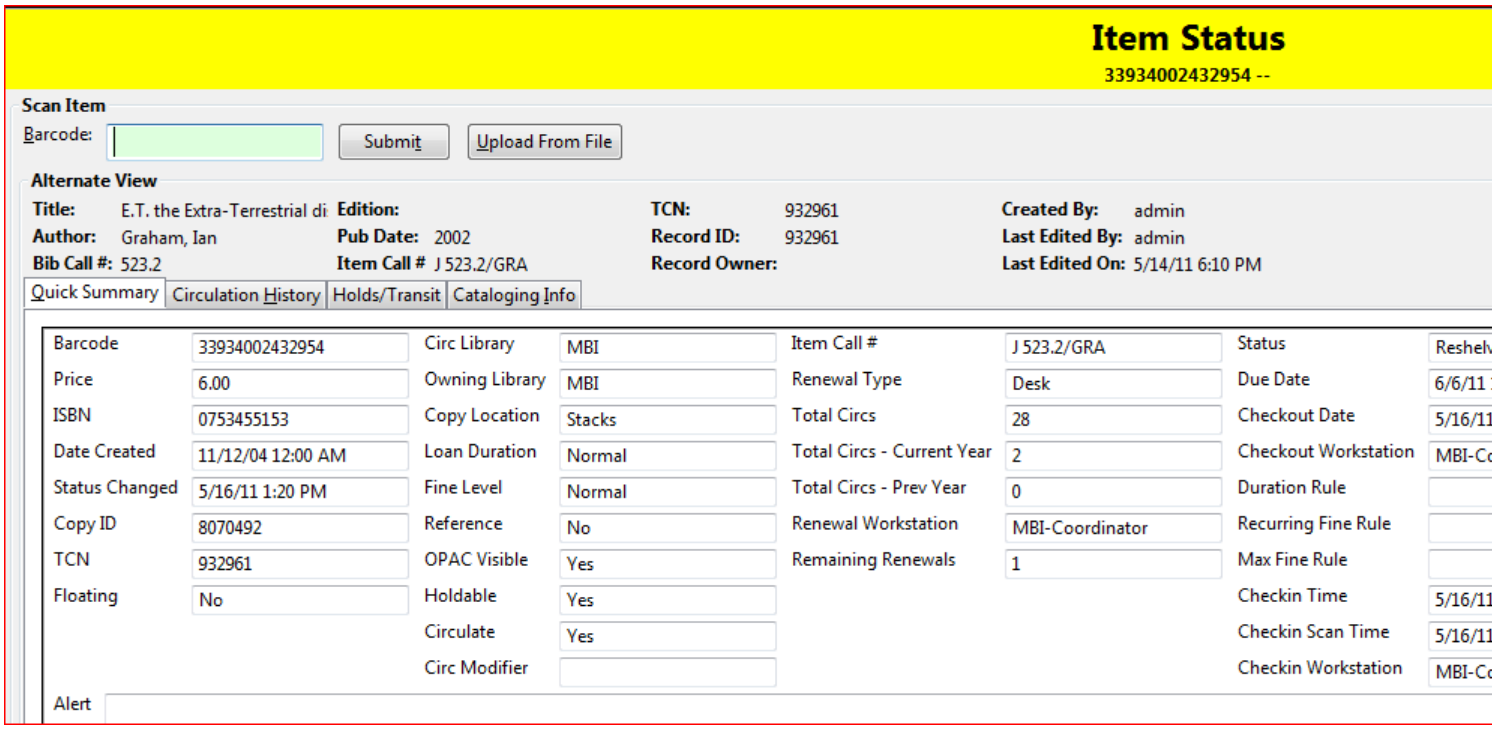
1) To check the status of an item, select **Search # Search for copies by Barcode** or **Circulation # Show Item Status by Barcode**; click the **Item Status button** on the circulation or cataloging toolbar; or press **F5**.



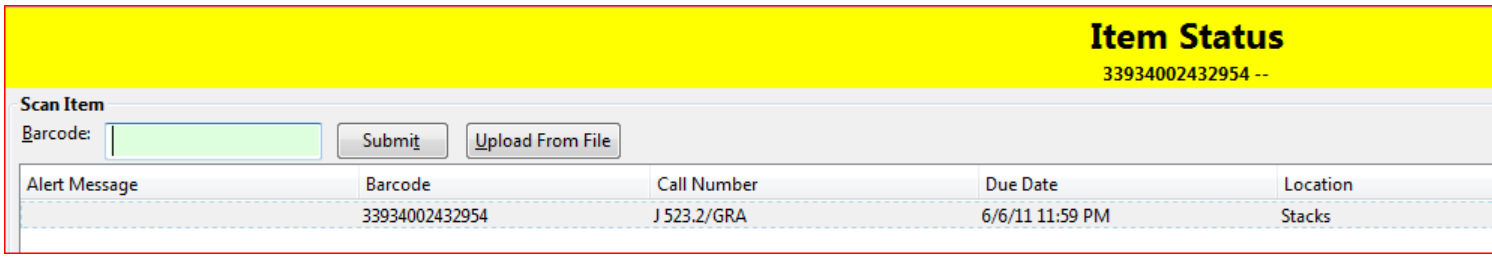
2) Scan the barcode or type it and click **Submit**. The current status of the item is displayed with selected other fields. You can use the column picker to select more fields to view.



3) Click the **Alternate View** button, and the item summary and circulation history will be displayed.



4) Click **List View** to go back.



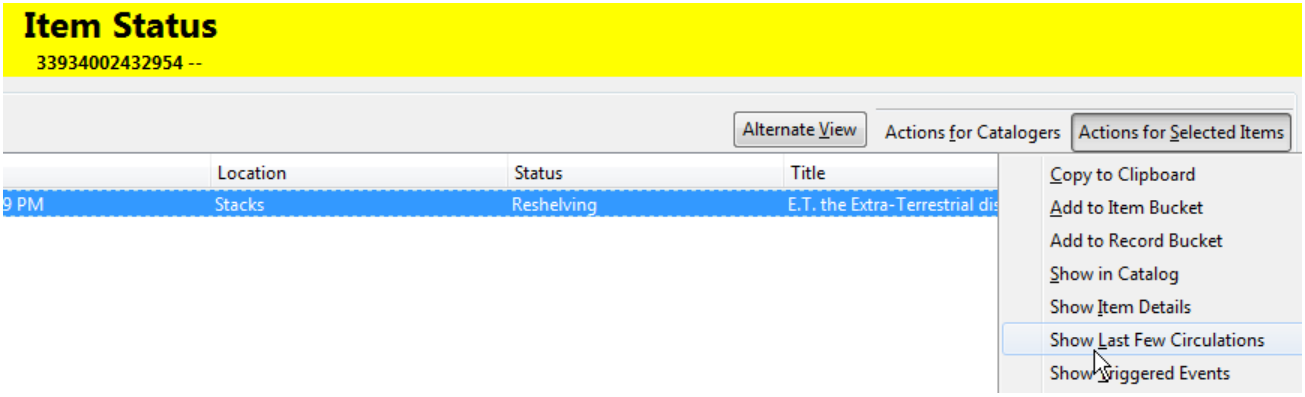
If the item's status is "Available", the displayed due date refers to the previous circulation's due date.

Upload From File allows you to load multiple items saved in a file on your local computer. The file contains a list of the barcodes in text format. To ensure smooth uploading and further processing on the items, it is recommended that the list contains no more than 100 items.

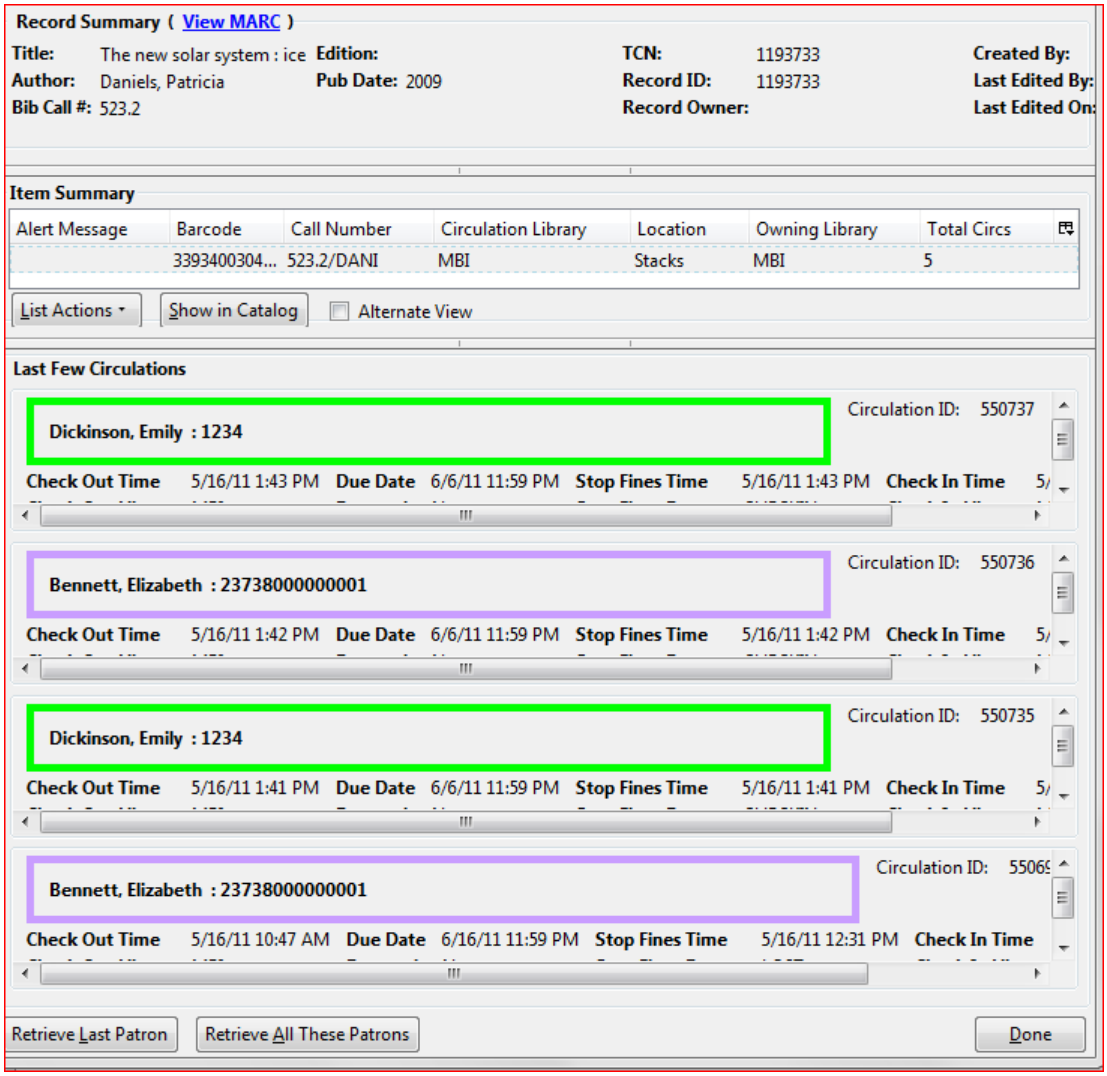
Viewing past circulations

1) To view past circulations, retrieve the item on the **Item Status** screen as described above.

2) Select **Actions for Selected Items # Show Last Few Circulations**.



3) The item's recent circulation history is displayed.



4) To retrieve the patron(s) of the last circulations, click the **Retrieve Last Patron** or the **Retrieve All These Patrons** button at the bottom of the above screen. Patron record(s) will be displayed in new tab(s).

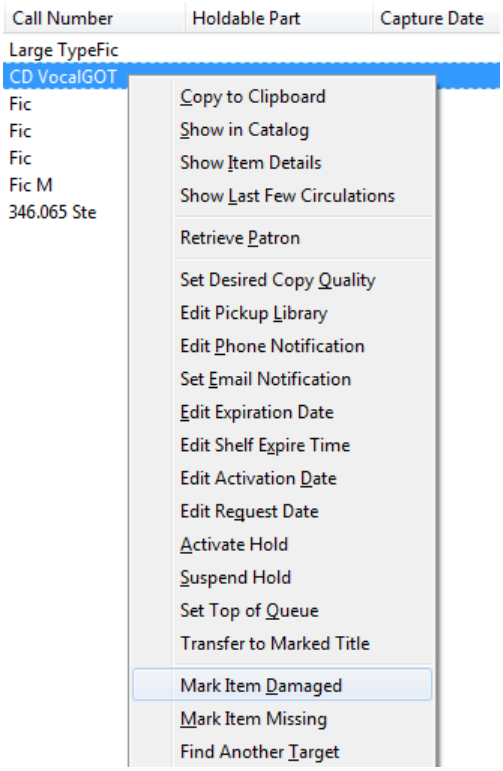
The number of items that displays in the circulation history can be set in Local **Administration** # **Library Settings Editor**.

You can also retrieve the past circulations on the patron's Items Out screen and from the Check In screen.

Marking items damaged or missing and other functions

1) To mark items damaged or missing, retrieve the item on the **Item Status** screen.

2) Select the item. Click on **Actions for Selected Items** # **Mark Item Damaged** or **Mark Item Missing**.



[NOTE} Depending on the library's policy, when marking an item damaged, bills (cost and/or processing fee) may be inserted into the last borrower's account.

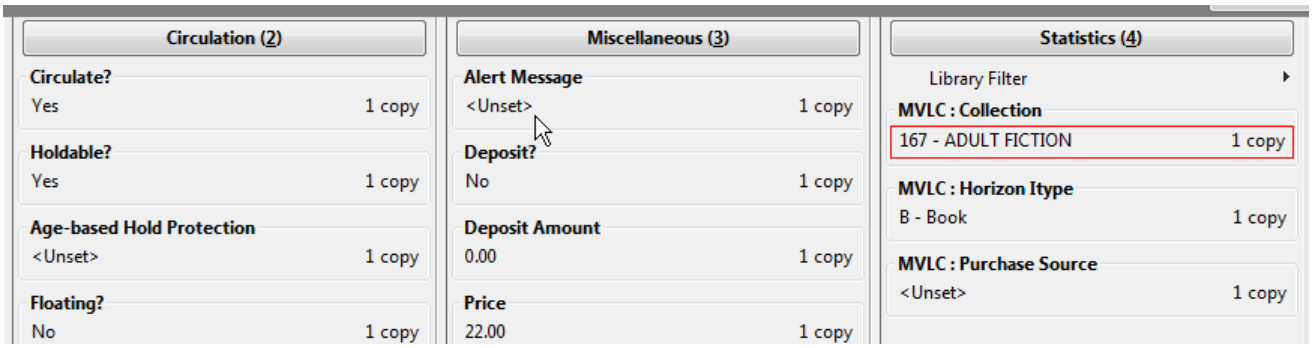
3) Following the above procedure, you can check in and renew items by using the **Check in Items** and **Renew Items** on the dropdown menu.

Item alerts

The **Edit Item Attributes** function on the **Actions for Selected Items** dropdown list allows you to edit item records. Here, we will show you how to insert item alert messages by this function. See cataloging instructions for more information on item editing. 1) Retrieve record on **Item Status** screen.

2) Once item is displayed, highlight it and select **Actions for Selected Items** # **Edit Item Attributes**.

3) The item record is displayed in the **Copy Editor**.



4) Click **Alert Message** in the **Miscellaneous** column. The background color of the box changes. Type in the message then click **Apply**.

5) Click **Modify Copies**, then confirm the action.

Holds

Placing Holds

Holds can be placed by staff in the Staff Client and by patrons in the OPAC. In this chapter we demonstrate placing holds on the Staff Client.

Holds Levels

Evergreen has five different levels of holds. Library staff can place holds at all five levels, while patrons can only place meta-record, title-level holds, and parts-level holds. The chart below summarizes the five levels of holds.

Table 4.1. Hold Levels

Hold level	Abbreviation	When to use	How to use	Who can use	Hold tied to
Meta-record	M	Patron wants first available copy of multiple titles of the same/ different format	Click on place hold next to the title. From holds confirmation screen, click Advanced hold	Patron or staff	Holdings attached to multiple MARC (title) records sharing the same title and author of

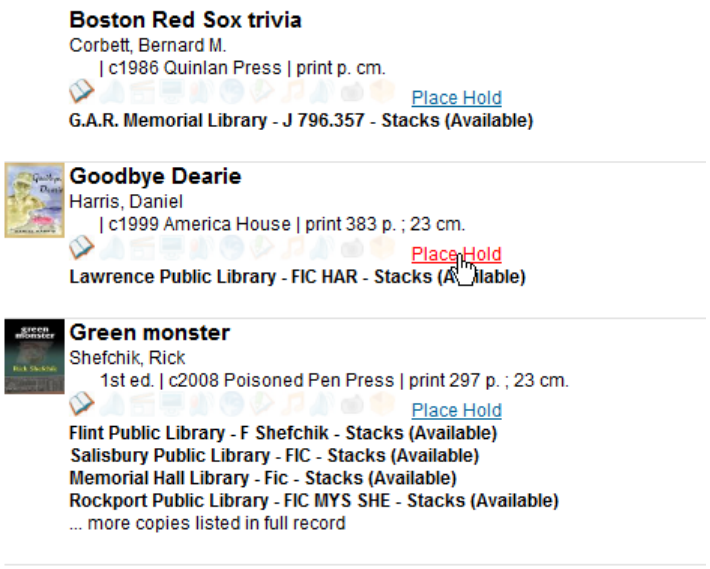
Hold level	Abbreviation	When to use	How to use	Who can use	Hold tied to
			options and select other applicable formats.		selected format(s) (book, video, audiobook, etc.)
Title	T	Patron wants first available copy of a title	Staff or patron click on place hold next to title.	Patron or staff	Holdings attached to a single MARC (title) record
Parts	P	Patron wants a particular part of title (e.g. volume or disk number)	Staff or patron selects part on the create/edit hold screen when setting holds notification options.	Patron or staff	Holdings with identical parts attached to a single MARC (title) record.
Volume	V	A call number specific volume of a title is required	Staff click place hold on any items shown in holdings list, next to the call number.	Staff	Holdings with identical call numbers owned by the same library and attached to a single MARC (title) record.
Copy	C	Patron or staff want a specific copy of a title	Staff click details to view barcode then place hold next to that barcode.	Staff	Item barcode

Title Level Hold

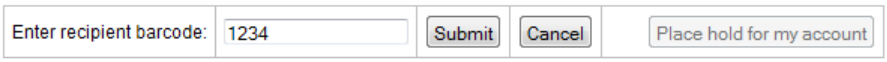
A default hold expiration date will be displayed if the library has set up a default holds expiration period in their library settings. Uncaptured holds will not be targeted after theexpiration date.

If you select the Suspend this Hold checkbox, the hold will be suspended and not be captured until you activate it.

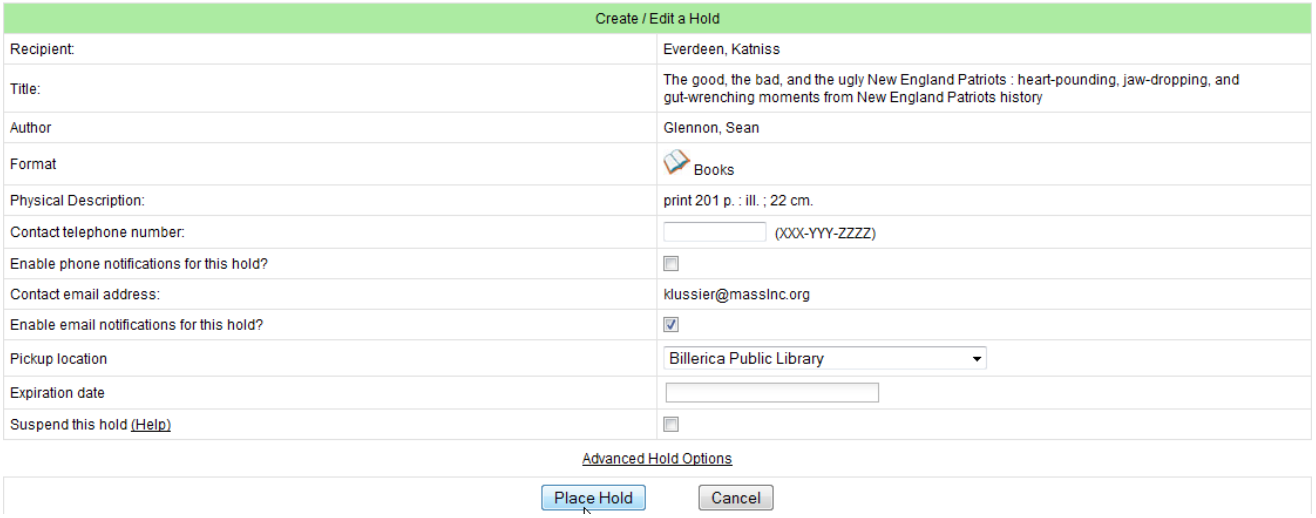
1. To place a title level hold, retrieve the title record on the catalog and click the Place Hold link beside the title on the search results list, or click the Place Hold link on the title summary screen.



2. Scan or type patron's barcode into Enter recipient barcode then click Submit.

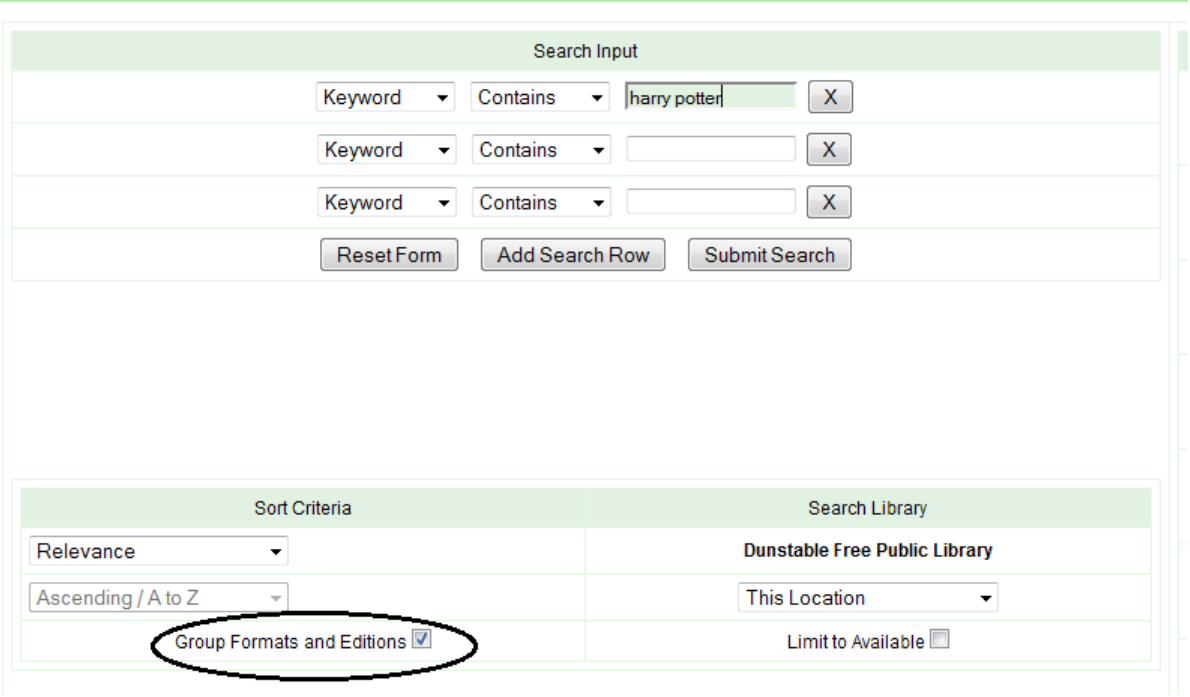


3. Edit patron hold notification and expiration date fields as required and click Place Hold. Confirm your action in the pop-up window.



Meta-record Level Hold

1. Search for the meta-record on which you want to place a hold. Select the Group by Formats and Editions checkbox from the advanced search screen. Enter search terms. Click Go.



2. Click Place Hold beside the meta-record on the result list.

> The lit-up icons indicate the system-wide available formats. Click the title to find out the available formats and holdings information at your library.

3. Type in or scan the patron barcode at the prompt.

4. Select the acceptable formats. Use the CTRL key together with a mouse click to select multiple entries. Click Place Hold once done.

> If you do not select group formats and editions when conducting a search, you can still place a hold on a meta-record. After entering the user's barcode, click Advanced Hold Options at the bottom of the screen to select acceptable formats for the hold.

5. After a meta-record hold is placed, if a new MARC record is added and it meets the grouping criteria (title, author and format), items under this new record will be used to fulfill the existing holds.

> Requested formats are listed in the Holdable Formats column in hold records. Use the Column Picker to display it when the hold record is displayed. Format information is from the MARC record leader.

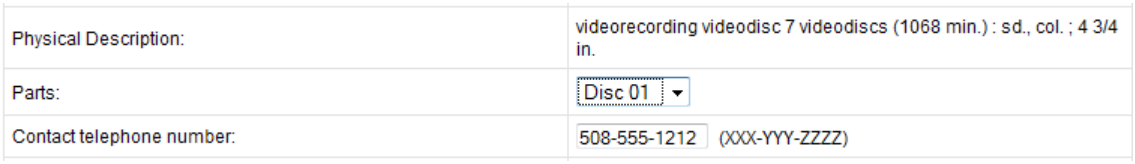
Parts Level Hold

1. To place a parts level, retrieve a record with parts-level items attached to the title, such as a multi-disc DVD, an annual travel guide, or a multi-volume book set.

2. Place the hold as you would for a title-level hold.

3. Scan or type patron's barcode into Enter recipient barcode then click Submit.

4. Select the applicable part from the Parts dropdown menu.



5. Click Place Hold once done.

Requested formats are listed in the Holdable Part column in hold records. Use the Column Picker to display it when the hold record is displayed.

Parts level holds cannot be placed on meta-records.

Volume Level Hold

1. To place a volume level hold, only possible within the Staff Client, retrieve and display the record.

2. Choose the appropriate volume record (call number). If not displayed, click View copy Information for all libraries to display all volumes.

3. Click Place Hold under Actions for the appropriate volume record. Your hold will be on any copy with the same call number and the same owning library.

4. Scan or type patron's barcode into Enter recipient barcode then click Submit.

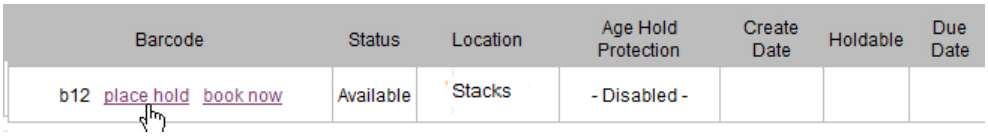
5. Edit patron hold notification fields as required and click Place Hold. Note the hold is identified as a Volume Hold.

Copy Level Hold

1. To place a copy level hold, repeat steps 1 and 2 in the section called Volume Level Hold.

2. Click on Copy details under Actions.

3. Click place hold beside the barcode. Your hold will be on this specific copy.

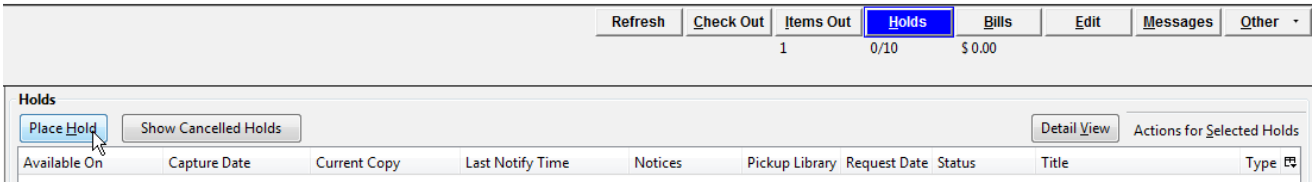


4. Scan or type patron's barcode into Enter recipient barcode then click Submit.

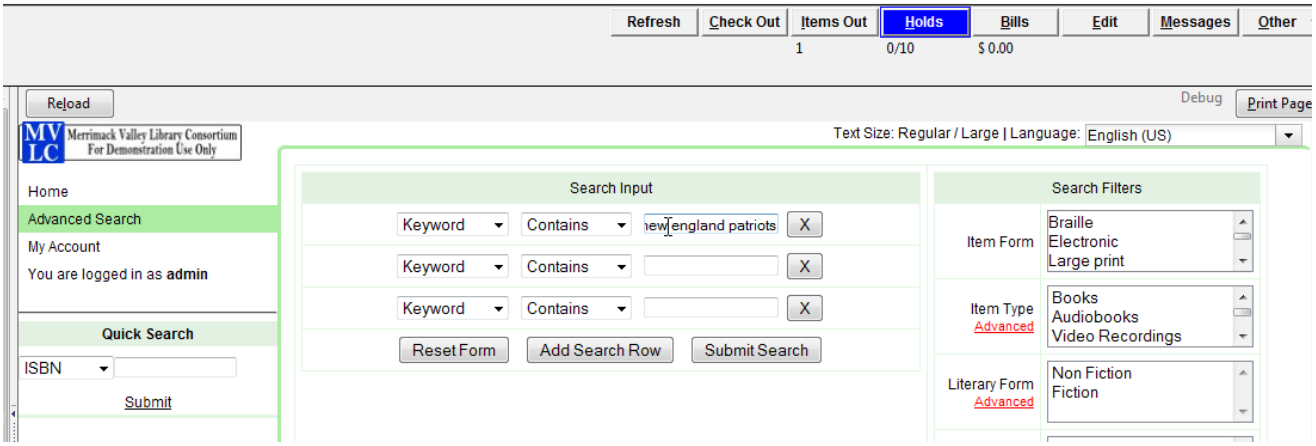
5. Edit the patron hold notification fields as required and click Place Hold. Note the hold is identified as a Copy Hold.

Placing Holds in Patron Records

1. Holds can be placed from patron records too. In the patron record Holds screen, click the Place Hold button on the left top corner.



2. The catalog is displayed in the Holds screen to search for the title on which you want to place a hold.



3. Search for the title and click the Place Hold link when you are at the appropriate place.

4. The patron's account information is retrieved automatically. Set up the notification and expiration date fields. Click Place Hold and confirm your action in the pop-up window.

5. You may continue to search for more titles. Once you are done, click the Holds button on top to go back to the Holds screen. Click the Refresh button to reflect your newly placed holds.

Managing Holds

Holds can be cancelled at any time by staff or patrons. Before holds are captured, staff or patrons can suspend them or set them as inactive for a period of time without losing the hold queue position, activate suspended holds, change notification method, phone number, pick-up location (for multi-branch libraries only), expiration date, activation date for inactive holds, etc. Once a hold is captured, staff can change the pickup location and extend the hold shelf time if required.

Staff can edit holds in either patron's records or the title records. Patrons can edit their holds in their account on the OPAC.

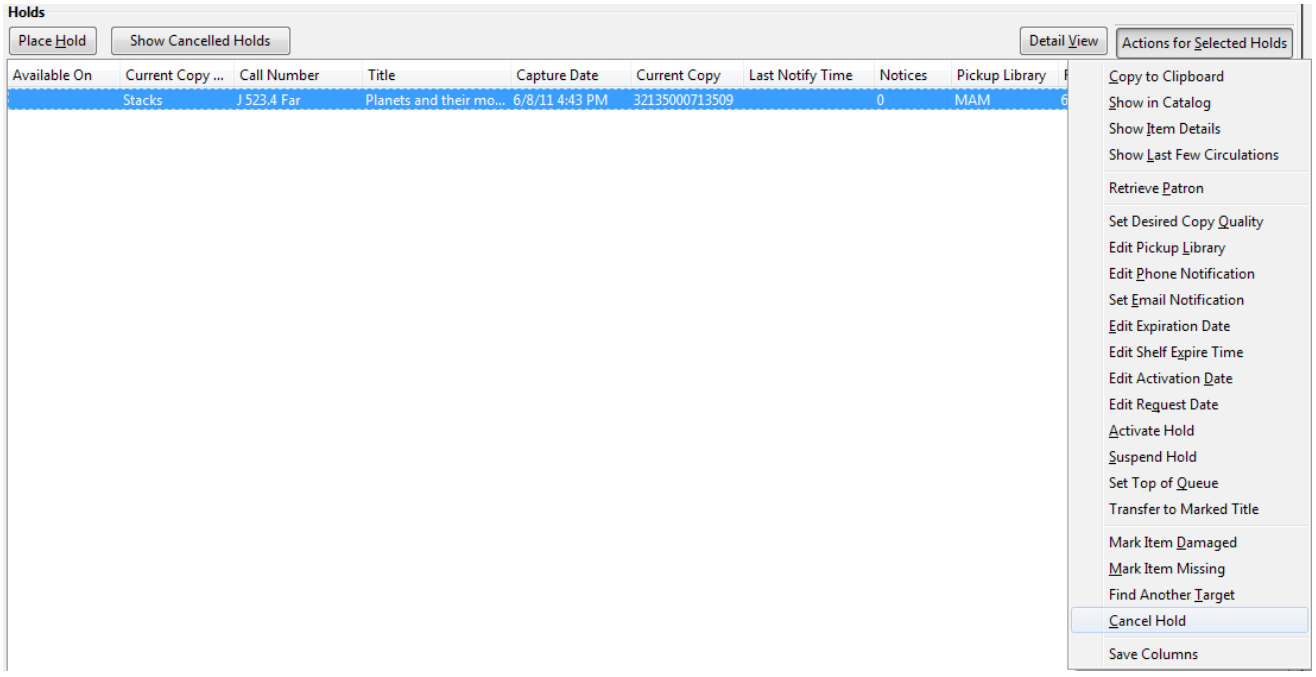
If you use the column picker to change the holds display from one area of the staff client (e.g. the patron record), it will change the display for all parts of the staff client that deal with holds, including the title record holds display, the holds shelf display, and the pull list display.

Managing Holds in Patron Records

Actions for Selected Holds

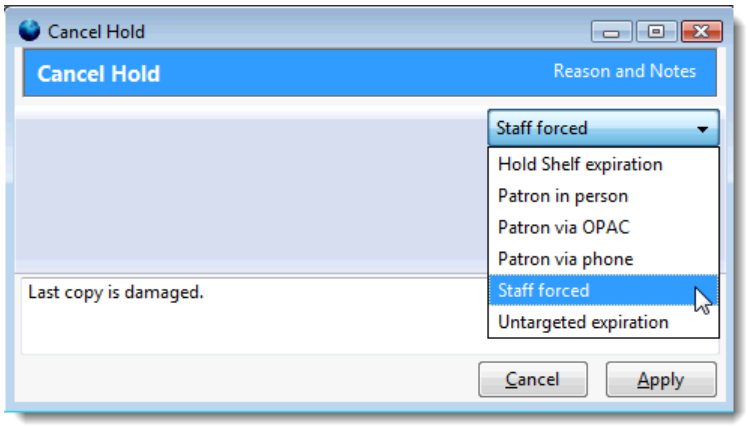
1. Retrieve the patron record and go to the Holds screen.

2. Highlight the hold record, then select Actions for Selected Holds.



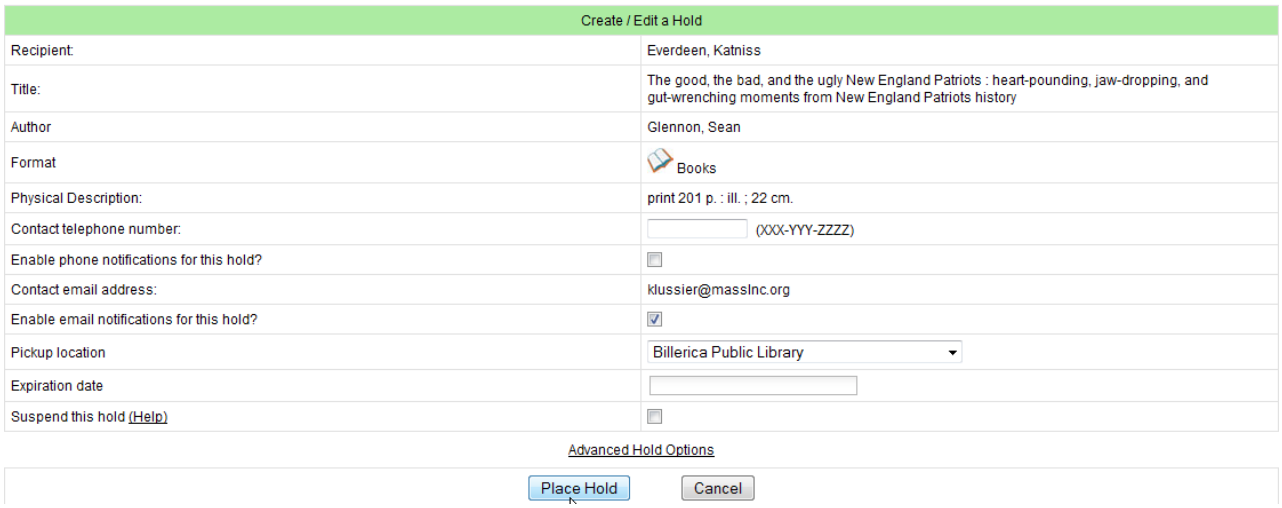
3. Manage the hold by choosing an action on the list.

   a. If you want to cancel the hold, click Cancel Hold on the above screen. You are prompted to select a reason and put in a note if required. Once done, click Apply.



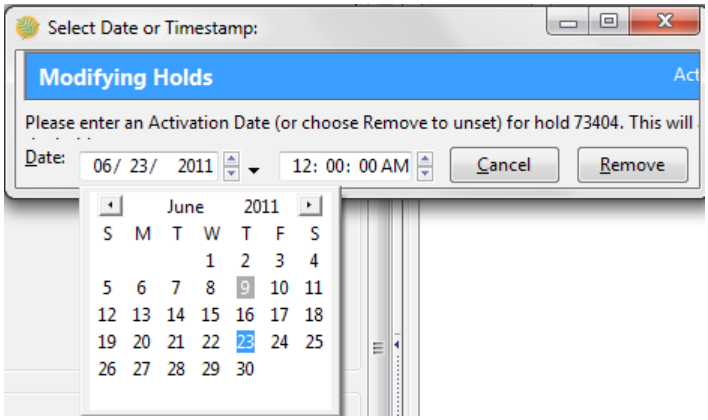
A captured hold with a status of "On Hold Shelf" can be cancelled by either staff or patrons. But the status of the item will not be changed until staff check it in.

b. If you want to suspend a hold or activate a suspended hold, click the appropriate action on the list. You will be prompted to confirm your action. Suspended holds have a `No` value in the Active? column.



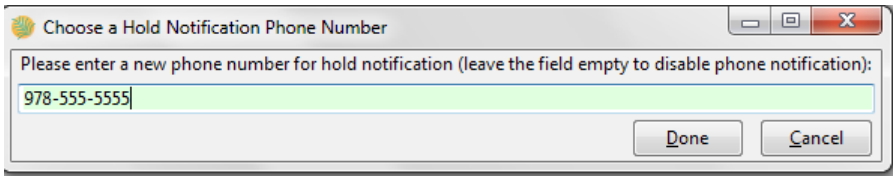
Suspended holds will not be filled but its hold position will be kept. They will automatically become active on the activation day if there is an activation date in the record. Without an activation date, the holds will remain inactive until staff or a patron activates them manually.

c. You may edit the Activation Date and Expiration Date by using the corresponding action entry on the Action for Selected Holds dropdown menu. You will be prompted to enter the new date. Use the calendar widget to choose a date, then click Apply. Use the Remove button to unset the date.

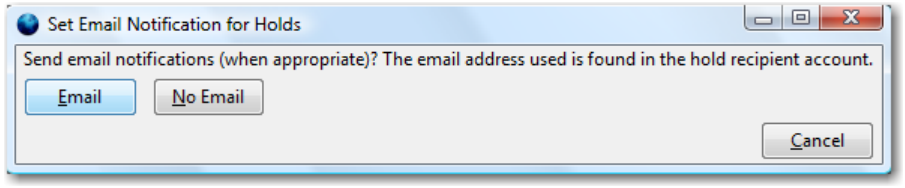


d. Hold shelf expire time is automatically recorded in the hold record when a hold is filled. You may edit this time by using the Edit Shelf Expire Time on the Action for Selected Holds dropdown menu. You will be prompted to enter the new date. Use the calendar widget to choose a date, then click Apply.

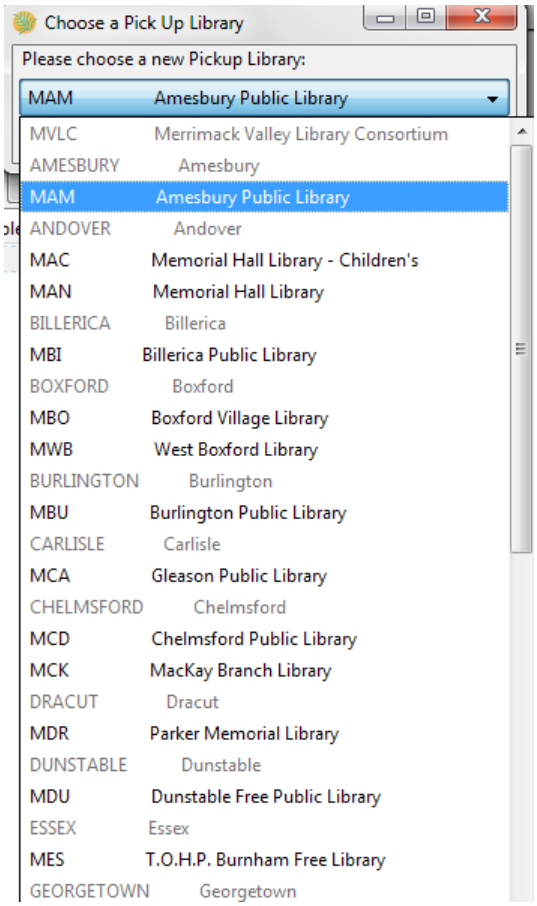
e. If you want to enable or disable phone notification or change the phone number, click Edit Phone Notification. You will be prompted to enter the new phone number. You must follow the format of XXX-XXX-XXXX. The phone number is used for this hold only and can be different from the one in the patron account. It has no impact on the patron account. If you leave it blank, no phone number will be printed on the hold slip.



f. If you want to enable or disable email notification for the hold, click Set Email Notification. Click Email or No Email on the prompt screen.

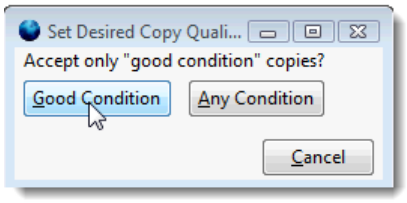


g. Pickup location can be changed by clicking Edit Pickup Library. Click the dropdown list of all libraries and choose the new pickup location. Click Done.



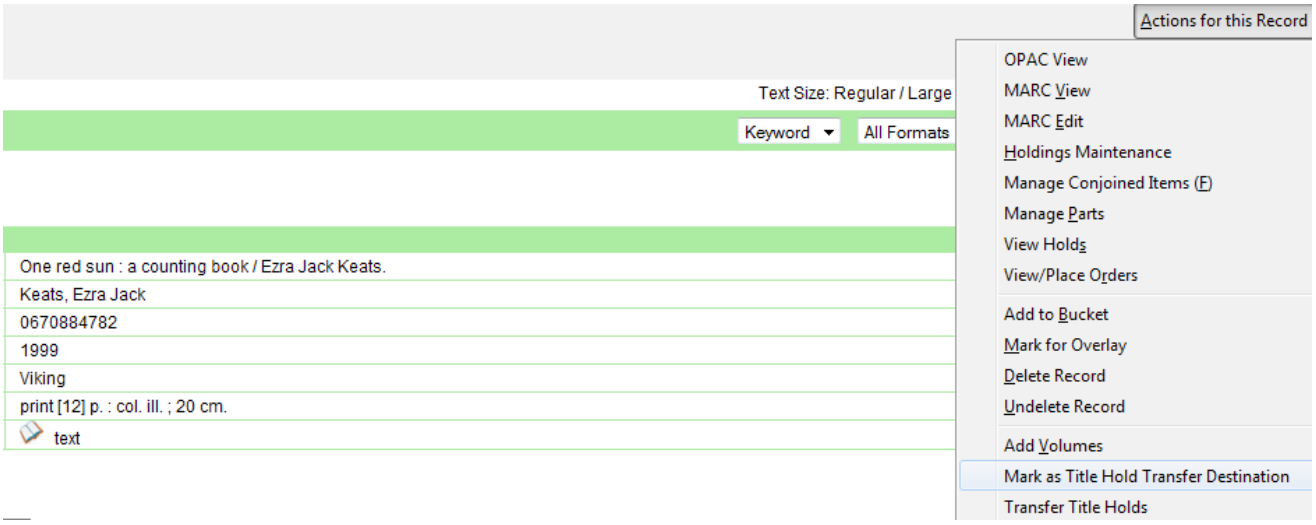
Staff can change the pickup location for holds with in-transit status. Item will be sent in transit to the new destination. Staff cannot change the pickup location once an item is on the holds shelf.

h. The item's physical condition is recorded in the copy record as Good or Mediocre in the Quality field. You may request that your holds be filled with copies of good quality only. Click Set Desired Copy Quality on the Actions for Selected Holds list. Make your choice in the pop-up window.

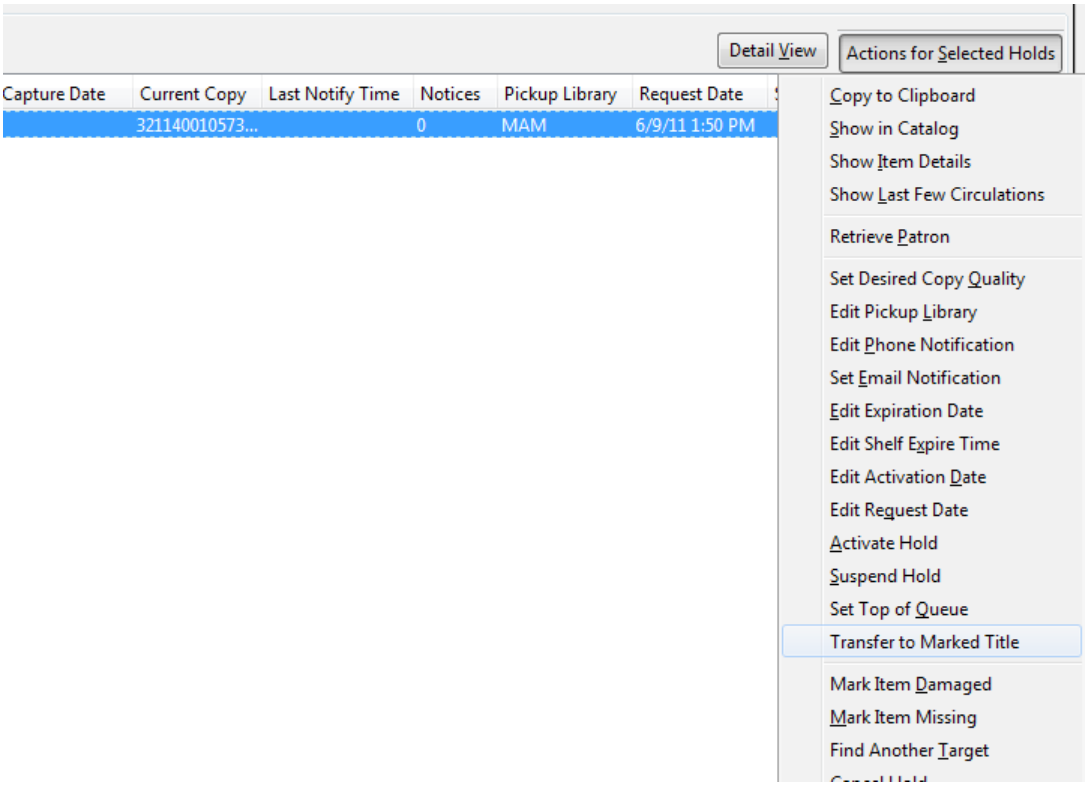


Transferring Holds

1. Holds on one title can be transferred to another with the hold request time preserved. To do so, you need to find the destination title, click Actions for this Record # Mark as Title Hold Transfer Destination.

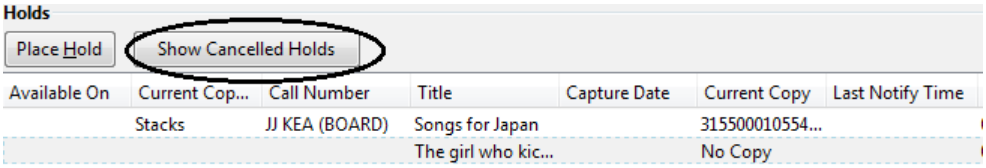


2. Select the hold you want to transfer. Click Actions for Selected Holds # Transfer to Marked Title.

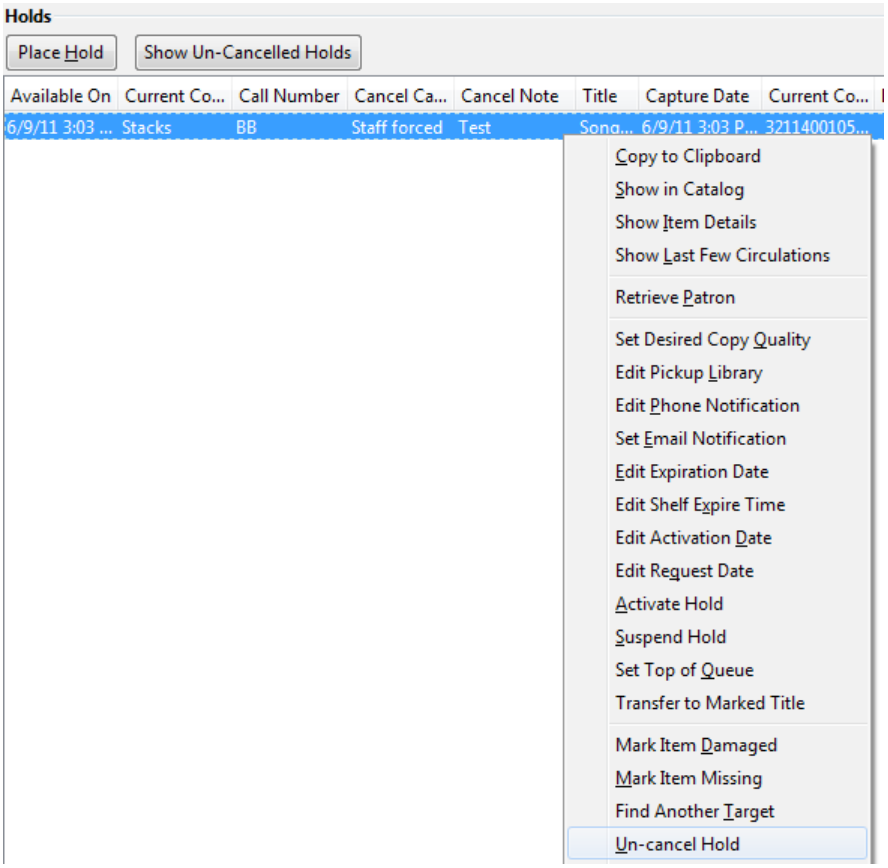


Cancelled Holds

1. Cancelled holds can be displayed. Click Show Cancelled Holds button on the Holds screen.



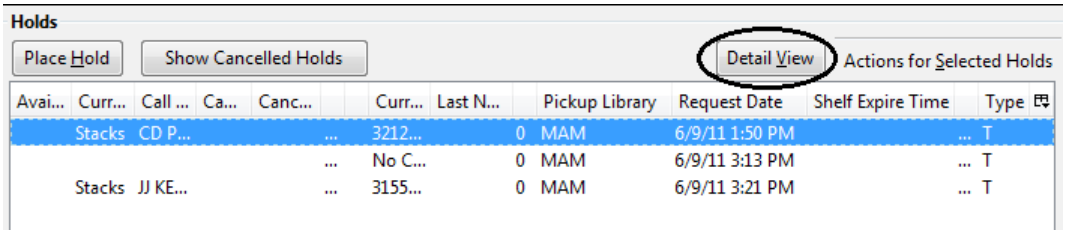
2. You can un-cancel holds.



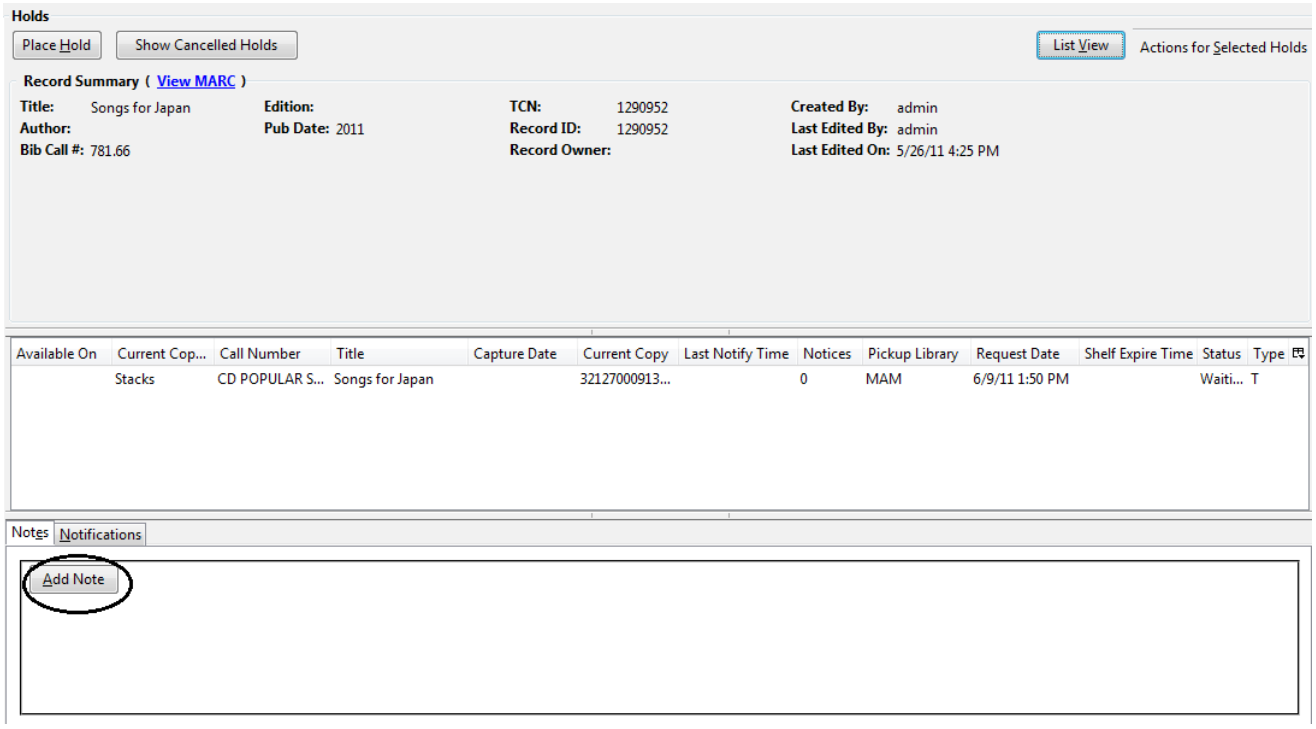
Based on your library's setting, hold request time can be reset when a hold is un-cancelled.

Viewing Details & Adding Notes to Holds

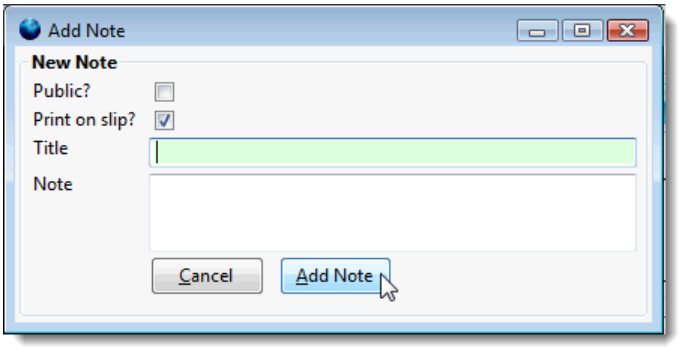
1. You can view details of a hold by selecting a hold then clicking the Detail View button on the Holds screen.



2. You may add a note to a hold in the Detail View.

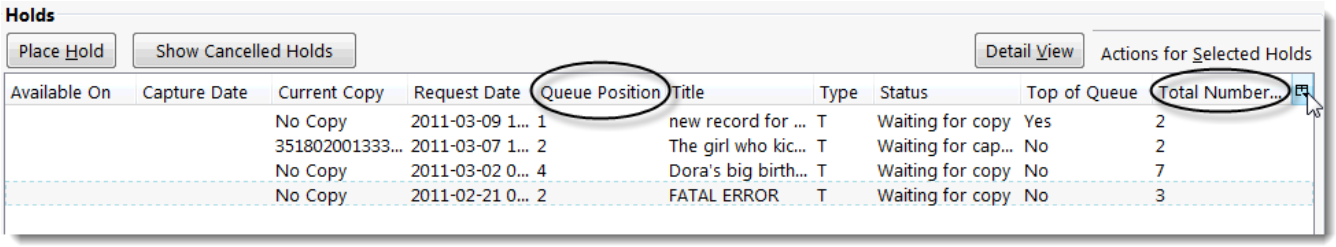


3. Notes can be printed on the hold slip if the Print on slip? checkbox is selected. Key in the message then click Add Note.



Displaying Queue Position

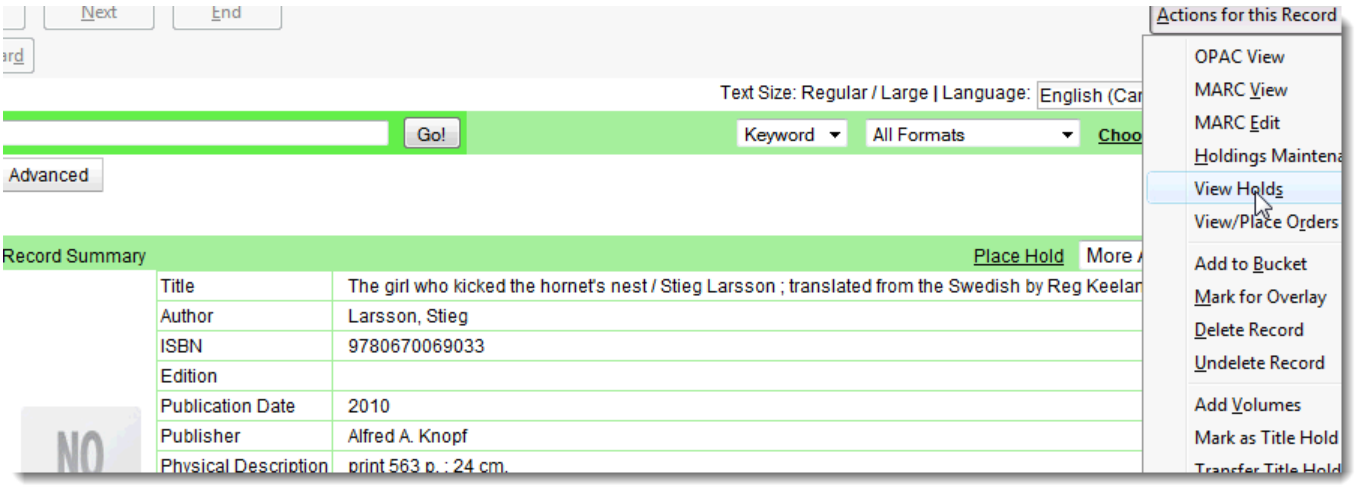
Using the Column Picker, you can display Queue Position and Total number of Holds.



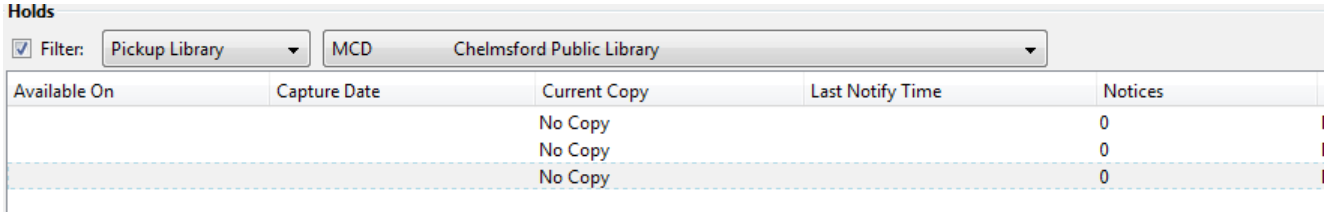
Managing Holds in Title Records

1. Retrieve and display the title record in the catalog.

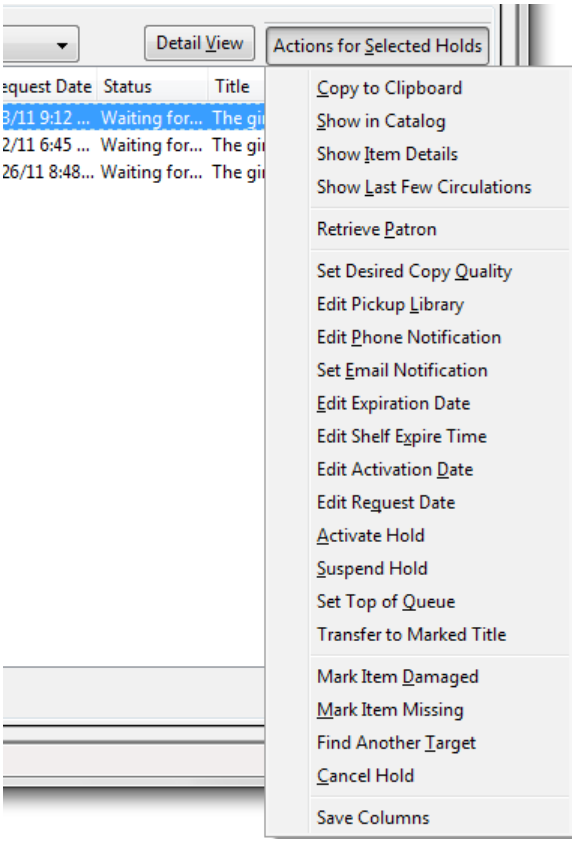
2. Click Actions for this Record # View Holds.



3. All holds on this title to be picked up at your library are displayed. Use Filter checkbox and Pickup Library to view holds to be picked up at other libraries.



4. Highlight the hold you want to edit. Click Actions for Selected Holds and the appropriate action you want to take as described in the Actions for Selected Holds section of Managing Holds in Patron Records.



5. You can retrieve the hold requestor's account by selecting Retrieve Patron on the above dropdown menu. Retargeting Holds

Holds need to be retargeted whenever a new item is added to a record, or after some types of item status changes, for instance when an item is changed from On Order to In Process. The system does not automatically recognize the newly added items as available to fill holds.

1. View the holds for the item.

2. Highlight all the holds for the record, which have a status of Waiting for Copy. If there are a lot of holds, it may be helpful to sort the holds by Status.

3. Click on the head of the status column.

4. Under Actions for Selected Holds (Alt+S) select Find Another Target (Alt+T)

5. A window will open asking if you are sure you would like to reset the holds for these items.

6. Click Yes (Alt+Y). Nothing may appear to happen, or if you are retargeting a lot of holds at once, your screen may go blank or seem to freeze for a moment while the holds are retargeted.

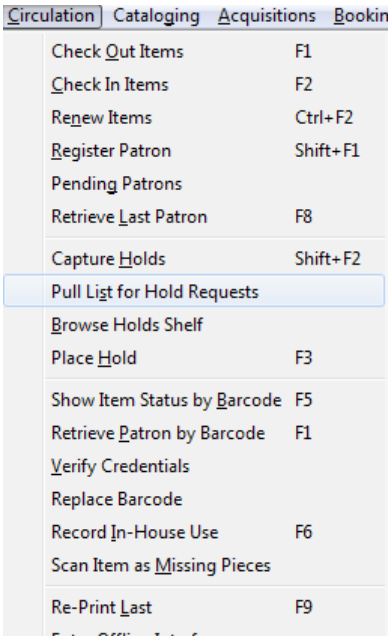
7. When the screen refreshes, the holds will be retargeted. The system will now recognize the new items as available for holds.

Pulling & Capturing Holds

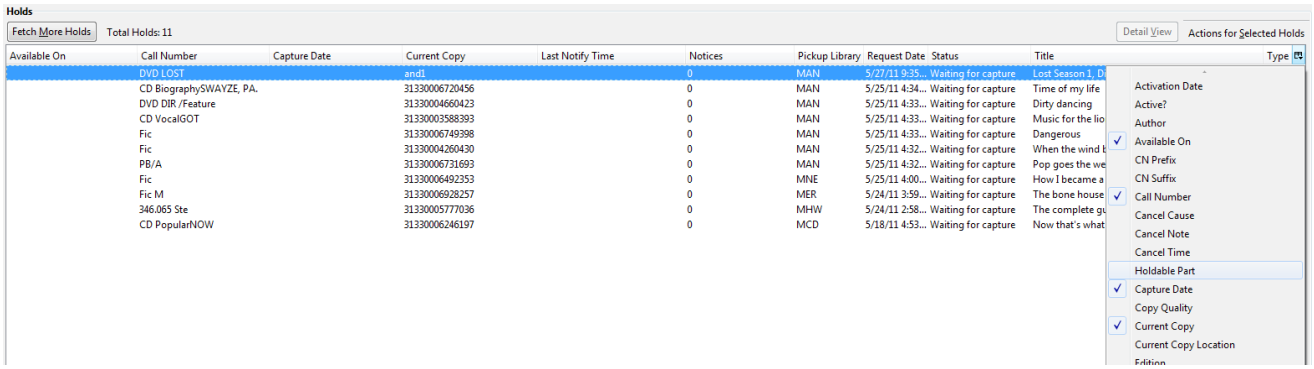
Holds Pull List

There are usually four types of status a hold may have: Waiting for Copy, Waiting for Capture, In Transit and Ready for Pickup.

1. Waiting-for-copy: all holdable copies are checked out or not available.
2. Waiting-for-capture: an available copy is assigned to the hold. The item shows up on the Holds Pull List waiting for staff to search the shelf and capture the hold.
3. In Transit: holds are captured at a non-pickup branch and on the way to the pick-up location.
4. Ready-for-pick-up: holds are captured and items are on the Hold Shelf waiting for patrons to pick up. Besides capturing holds when checking in items, Evergreen matches holds with available items in your library at regular intervals. Once a matching copy is found, the item's barcode number is assigned to the hold and the item is put on the Holds Pull List. Staff can print the Holds Pull List and search for the items on shelves.
5. To retrieve your Holds Pull List select Circulation # Pull List for Hold Requests.

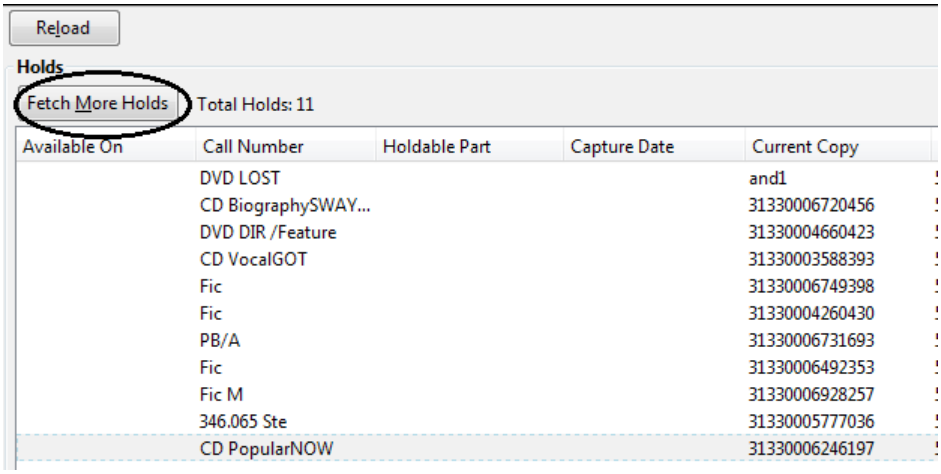


6. The Holds Pull List is displayed. You may re-sort it by clicking the column labels, e.g. Title. You can also add fields to the display by using the column picker.



Column adjustments will only affect the screen display and the CSV download for the holds pull list. It will not affect the printable holds pull list.

7. The maximum number of holds initially displayed on the pull list is about 100. Use Fetch More Holds to retrieve more records. You may have to click Reload for those records to appear in the display.

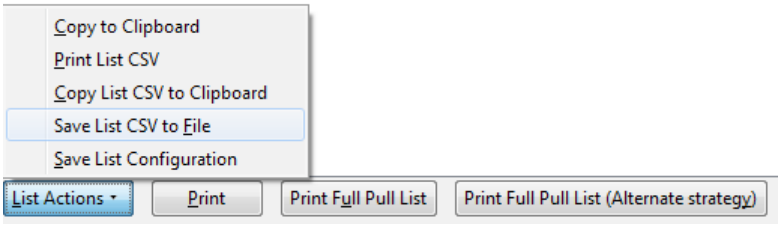


8. The following options are available for printing the pull list:

   - Print Full Pull List prints Title, Author, Shelving Location, Call Number and Item Barcode. This method uses less paper than the alternate strategy.

   - Print Full Pull List (Alternate Strategy) prints the same fields as the above option but also includes a patron barcode. This list will also first sort by copy location, as ordered under Admin#Local Administration#Copy Location Order.

   - Save List CSV to File – This option is available from the List Actions button and saves all fields in the screen display to a CSV file. This file can then be opened in Excel or another spreadsheet program. This option provides more flexibility in identifying fields that should be printed.



   With the CSV option, if you are including barcodes in the holds pull list, you will need to take the following steps to make the barcode display properly in Excel: select the entire the barcode column, right-click and select Format Cells, click Number as the category and then reduce the number of decimal places to 0.

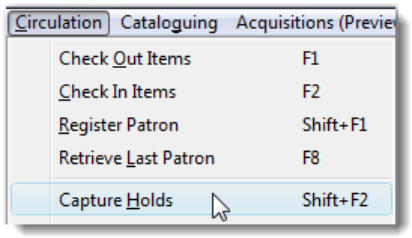
9. You may perform hold management tasks by using the Actions for Selected Holds dropdown list.

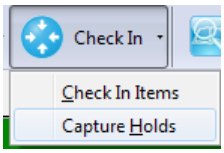
The Holds Pull List is updated constantly. Once an item on the list is no longer available or a hold on the list is captured, the items will disappear from the list. The Holds Pull List should be printed at least once a day.

Capturing Holds

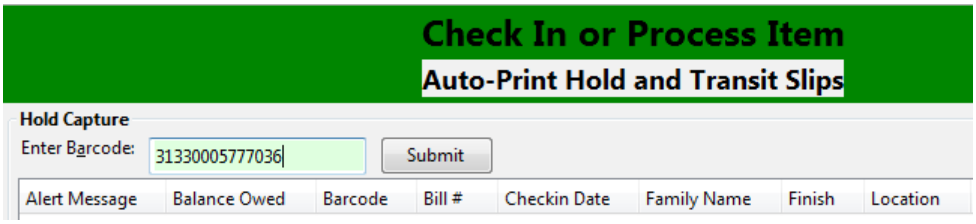
Holds can be captured when a checked-out item is returned (checked in) or an item on the Holds Pull List is retrieved and captured. When a hold is captured, the hold slip will be printed and if the patron has chosen to be notified by email, the email notification will be sent out. The item should be put on the hold shelf.

1. To capture a hold, Select Circulation -> Capture Holds; click Check In -> Capture Holds on the circulation toolbar; or hit Shift-F2.

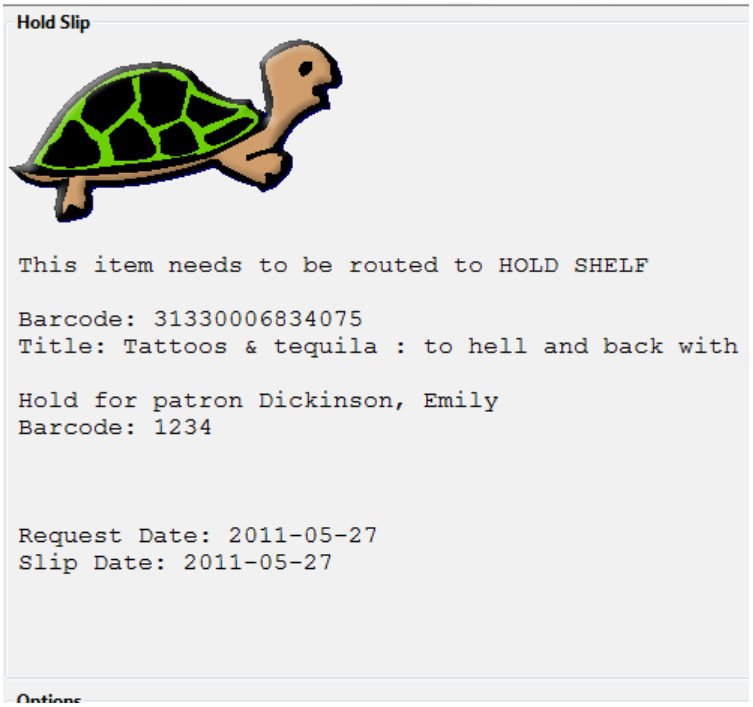




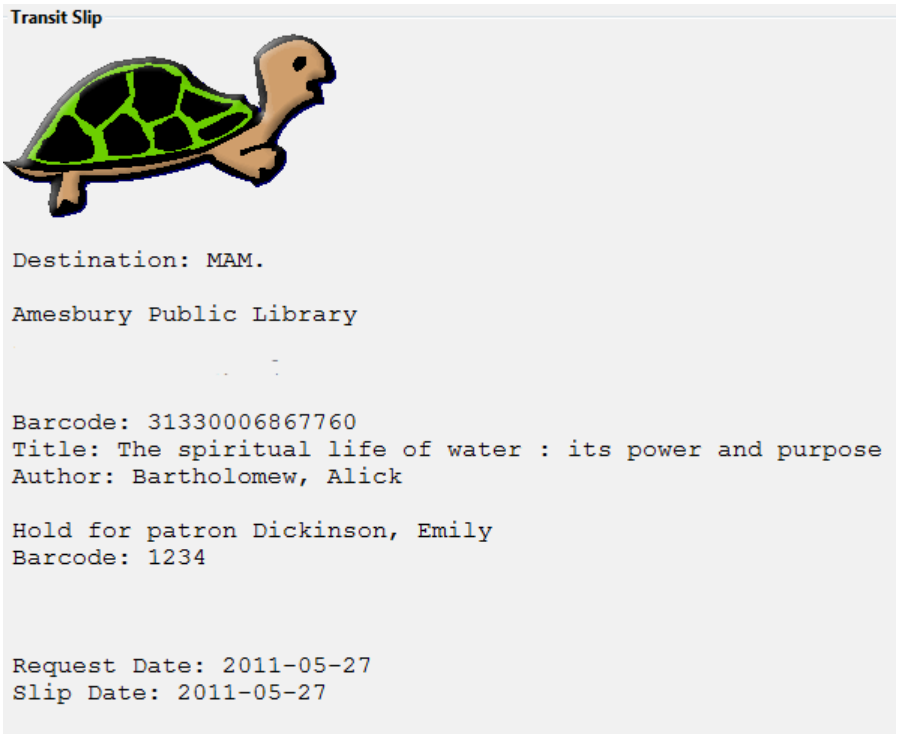
2. Scan or type barcode and click Submit.



3. The following hold slip is automatically printed. (This slip will not display on the Capture Holds screen, but will display on a Check In screen not set to automatically print slips.)



4. If the item should be sent to another location, a hold transit slip will be printed. (This slip will not display on the screen the Capture Holds screen, but may display on a Check In screen that is not set to automatically print slips.)

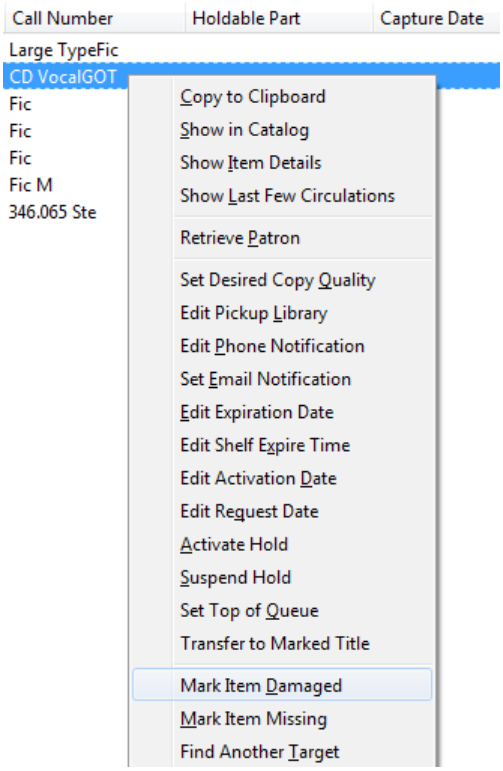


If a patron has an OPAC/Staff Client Holds Alias in his/her account, it will be used on the hold slip instead of the patron's name. Holds can also be captured on the Circulation # Check In Items screen where you have more control over automatic slip printing. Handling Missing and Damaged Items

Handling Missing and Damaged Items

If an item on the holds pull list is missing or damaged, you can change its status directly from the holds pull list.

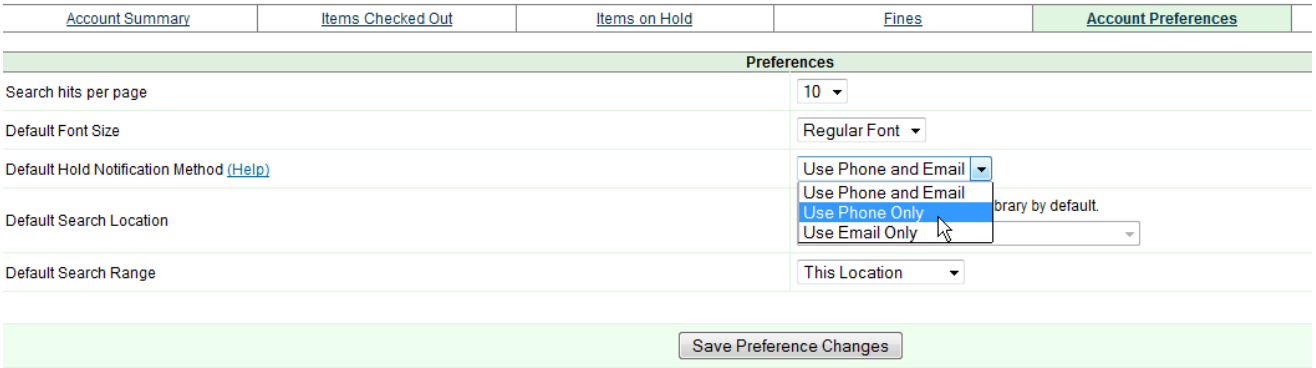
1. From the Holds Pull List, right-click on the item and either select Mark Item Missing or Mark Item Damaged.



2. Evergreen will update the status of the item and will immediately retarget the hold.

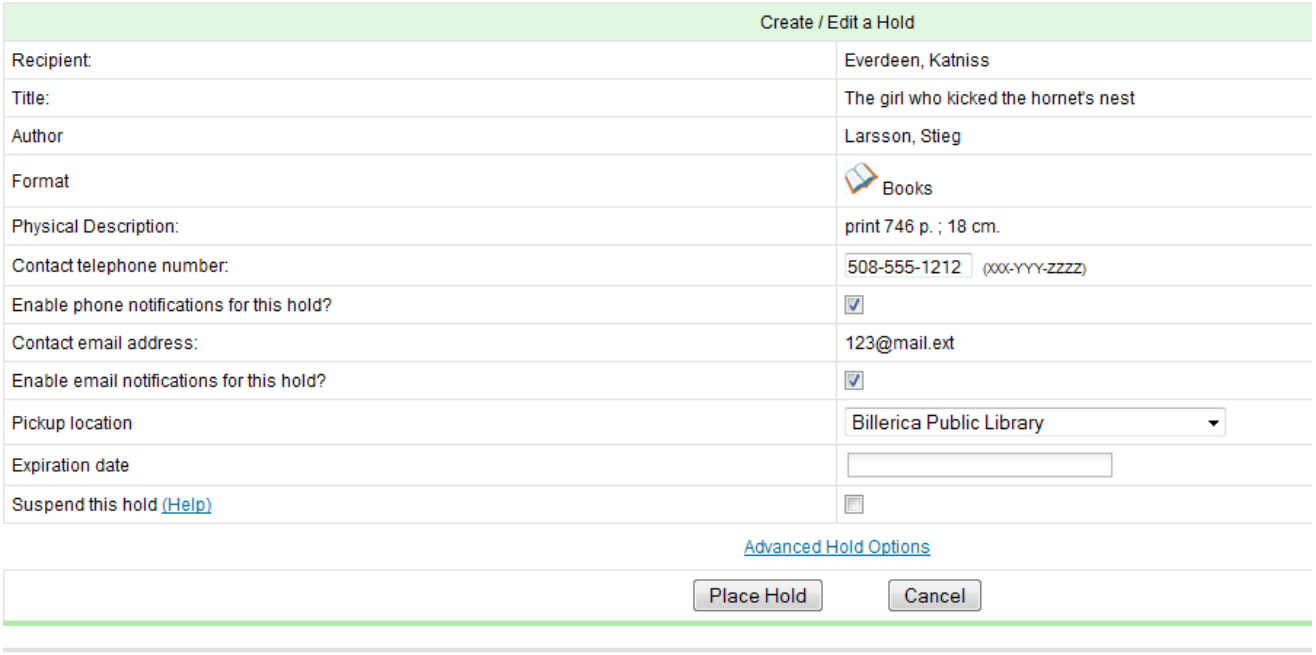
Holds Notification Methods

1. In Evergreen, patrons can set up their default holds notification method in the Account Preferences area of My Account. Staff cannot set these preferences for patrons; the patrons must do it when they are logged into the public catalog.

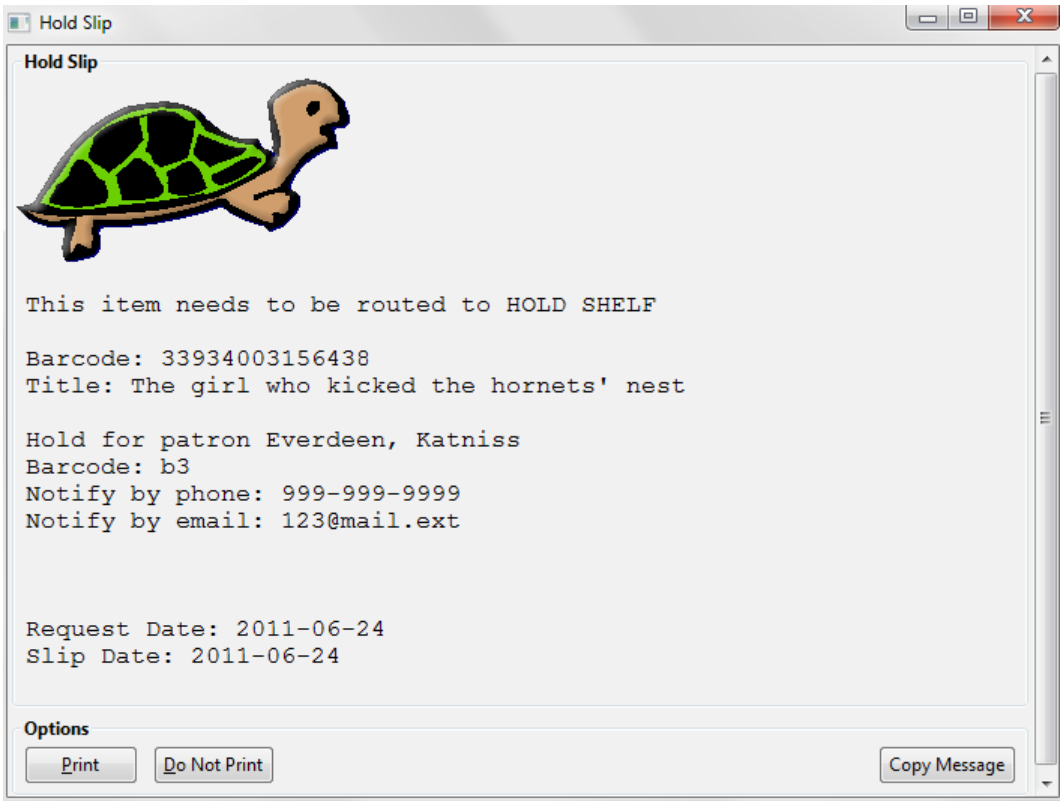


The “phone and email” option is the default for those users who have not yet set a preference.

2. Patrons with a default notification preference for phone and e-mail will see their phone number and e-mail address at the time they place a hold. The checkboxes for email and phone notification will also automatically be checked.



3. The patron can remove these checkmarks at the time they place the hold or they can enter a different phone number if they prefer to be contacted at a different number. The patron cannot change their e-mail address at this time.



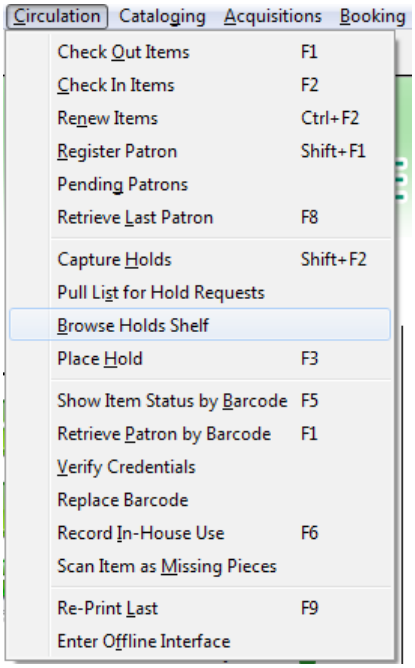
4. When the patron's hold becomes available, the system will only send an email if the Enable email notifications for this hold? checkbox was selected at the time the hold was placed.

5. When the hold becomes available, the holds slip will display the patron's e-mail address only if the patron selected the Enable email notifications for this hold? checkbox. It will display a phone number only if the patron selected the Enable phone notifications for this hold? checkbox.

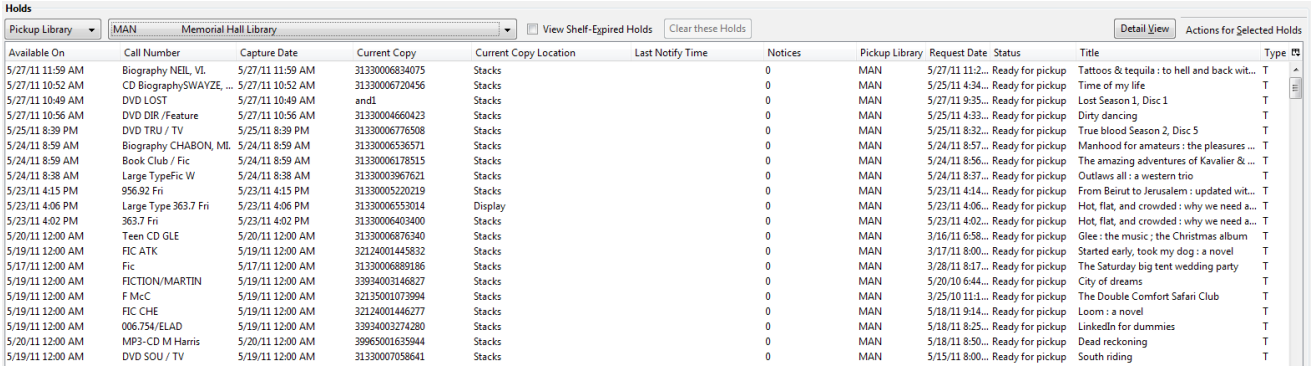
If the patron changes their contact telephone number when placing the hold, this phone number will display on the holds slip. It will not necessarily be the same phone number contained in the patron's record.

Clearing Shelf-Expired Holds

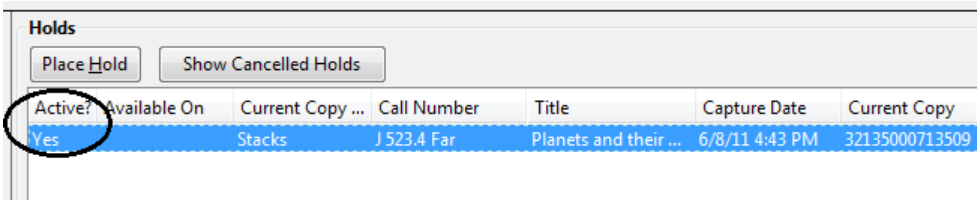
1. Items with Ready-for-pickup status are on the Holds Shelf List. The Holds Shelf List can help you manage items on the holds shelf. To see the holds shelf list, select Circulation -> Browse Holds Shelf.



2. The Holds Shelf List is displayed. Note the Actions for Selected Holds are available, as in the patron record. You can cancel stale holds here.

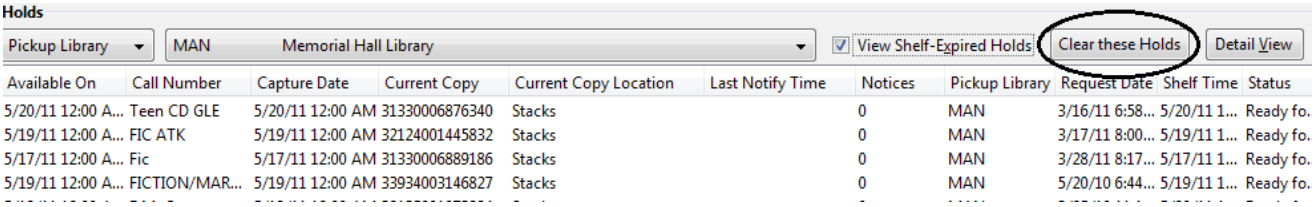


3. Use the column picker to add and remove fields from this display. Two fields you may want to display are Shelf Expire Time and Shelf Time.



4. Check the View Shelf-Expired Holds checkbox to list expired holds only. Click the Print button if you need a printed list.

5. The Clear These Holds button is lit up. Click it and the expired holds will be canceled.



6. Bring items down from the hold shelf and check them in.

If you cancel a ready-for-pickup hold, you must check in the item to make it available for circulation or trigger the next hold in line.

Hold shelf expire time is inserted when a hold achieves on-hold-shelf status. It is calculated based on the interval entered in Local Admin# Library Settings # Default hold shelf expire interval.

The clear-hold-shelf function cancels shelf-expired holds only. It does not include holds canceled by patron. Staff needs to trace these items manually according to the hold slip date.

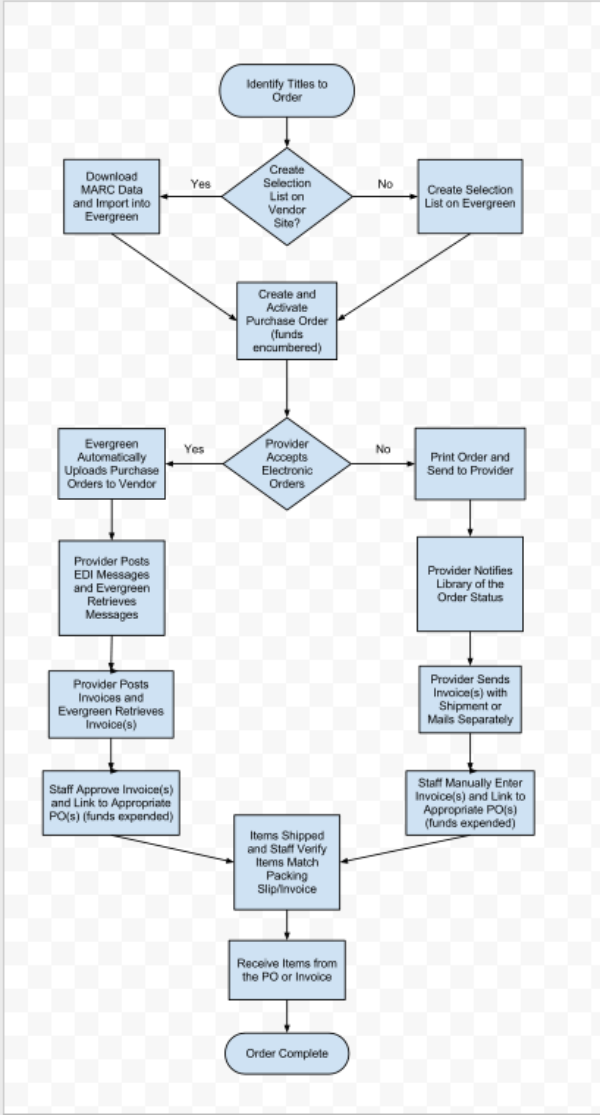
Chapter 5. The Acquisitions Module

Before beginning to use Acquisitions, the following must be configured by an administrator:

- Currency Types (defaults exist)
- Exchange Rates (defaults exist)
- Funds
- Providers
- EDI Accounts (optional)
- Claiming (optional)
- Invoices
- Distribution Formulas (optional)
- Line Item Features (optional)
- Cancel/Suspend Reasons (optional)
- See the section on administrative functions in the acquisitions module for details on acquisitions setup.

Acquisitons Workflow

The following diagram shows how the workflow functions in Evergreen. One of the differences in this process you should notice is that when creating a selection list on the vendor site, libraries will be downloading and importing the vendor bibs and item records.

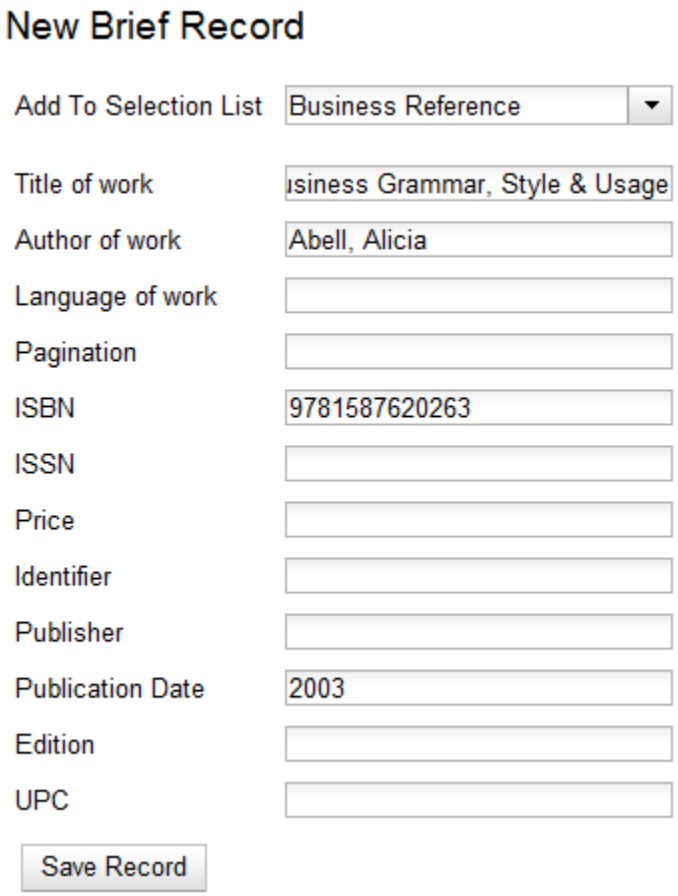


Brief Records

Brief records are short bibliographic records with minimal information that are often used as placeholder records until items are received. Brief records can be added to selection lists or purchase orders and can be imported into the catalog. You can add brief records to new or existing selection lists. You can add brief records to new, pending or on-order purchase orders.

Add brief records to a selection list

1. Click Acquisitions → New Brief Record. You can also add brief records to an existing selection list by clicking the Actions menu on the selection list and choosing Add Brief Record.
2. Choose a selection list from the drop down menu, or enter the name of a new selection list.
3. Enter bibliographic information in the desired fields.
4. Click Save Record.



Add brief records to purchase orders

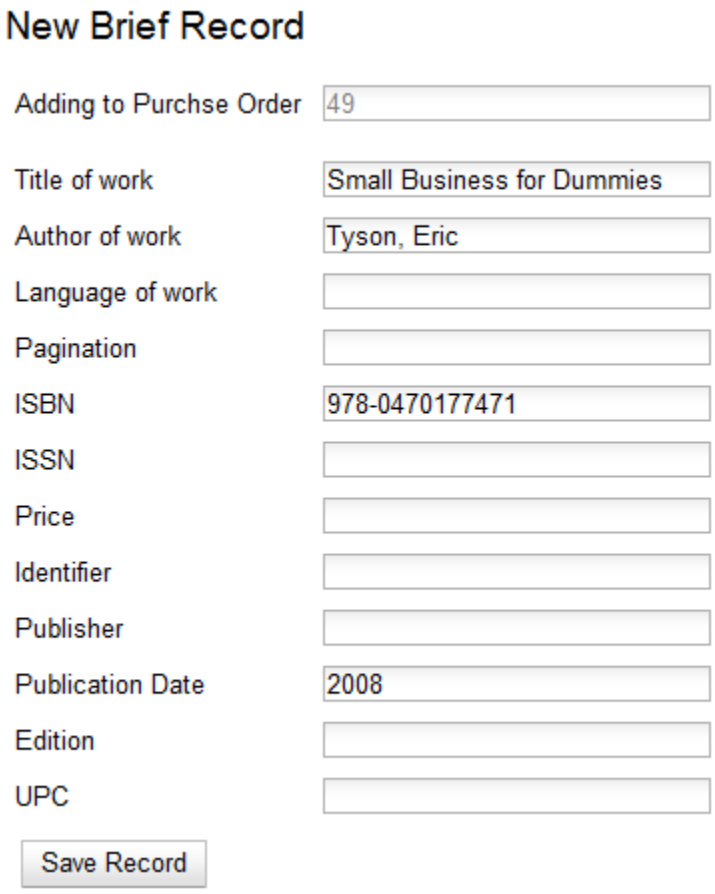
You can add brief records to new or existing purchase orders.

1. Open or create a purchase order. See the section on purchase orders for more information.

2. Click Add Brief Record.

3. Enter bibliographic information in the desired fields. Notice that the record is added to the purchase order that you just created.

4. Click Save Record.



Cancel/suspend acquisitions

You can cancel entire purchase orders, line items on the purchase orders, and individual copies that are attached to a line item. You can also use cancel reasons to suspend purchase orders, line items, and copies. For example, a cancel reason such as Delayed Publication, would indicate that the item will be purchased when the item is published. The purchase is, in effect, suspended rather than cancelled, but the state of the purchase order, line item, or copy would still become cancelled.

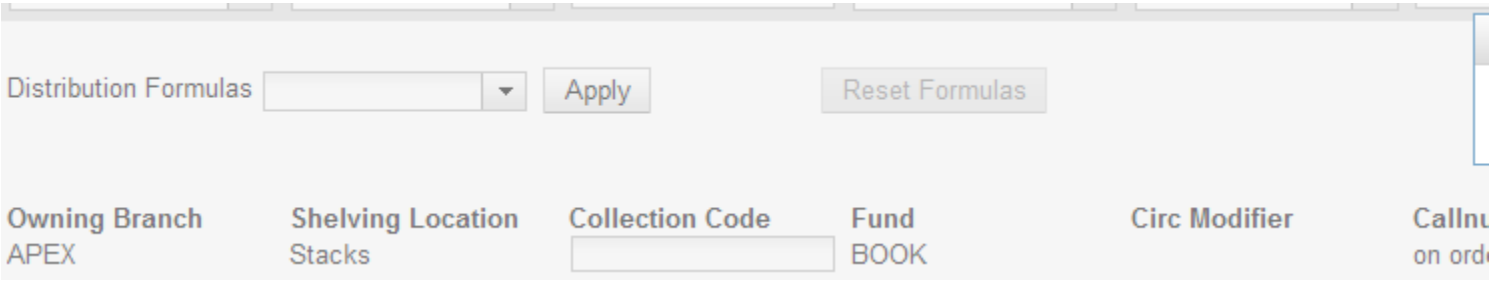
Cancel/suspend copies

You can cancel or suspend line items that are in a state of on order or pending order.

1. Select the Copies link.

2. Click the Cancel link adjacent to the copy that you wish to cancel.

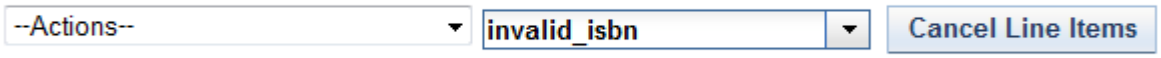
3. Select a cancel reason from the drop down menu that appears, and click Cancel copy.



Cancel/suspend line items

You can cancel or suspend line items that are in a state of on order or pending order.

1. Check the boxes of the line items that you wish to cancel.
2. Click Actions → Cancel Selected Lineitems.
3. Select a cancel reason from the drop down menu. Choose the cancel reason, and click Cancel Line Items. The status of the line item is now cancelled.



Cancel/suspend purchase orders

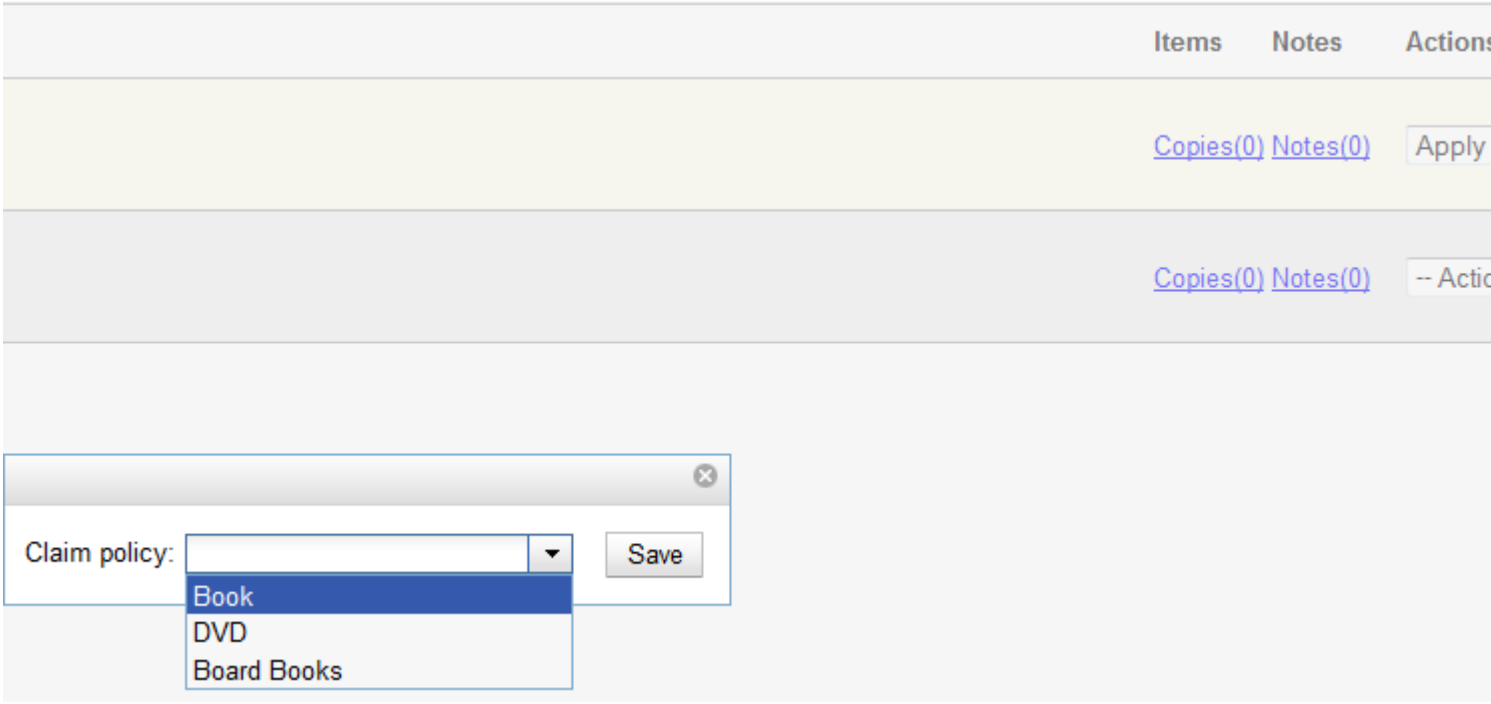
1. Notice the Cancel column in the top half of the purchase order.
2. Click the drop down arrow adjacent to Cancel order, and select a reason for cancelling the order.
3. Click Cancel order. The state of the purchase order is cancelled.

Claim items

Manual claiming of items can be accomplished in multiple ways, but electronic claiming is not available. You can apply claim policies to line items or individual copies. You also can use the default claim policy associated with your provider to claim items.

Apply a claim policy

You can apply a claim policy to an item in one of two ways: apply a claim policy to a line item when the item is created on the selection list or purchase order, or use the default claim policy associated with the provider on the purchase order. The default claim policy for a provider is established when the provider is created and will be used for claiming if no claim policy has been applied.



1. Open a selection list or purchase order. See the section on Purchase Orders for more information.
2. Click the Actions drop down menu on the line item.
3. Click Apply Claim Policy.
4. A drop down menu of claim policies will appear. Choose a claim policy to apply to the line item. The claim policy will be applied to all items that have not been received or cancelled.
5. Click Save.

Change a claim policy

You can manually change a claim policy that has been applied to a line item.

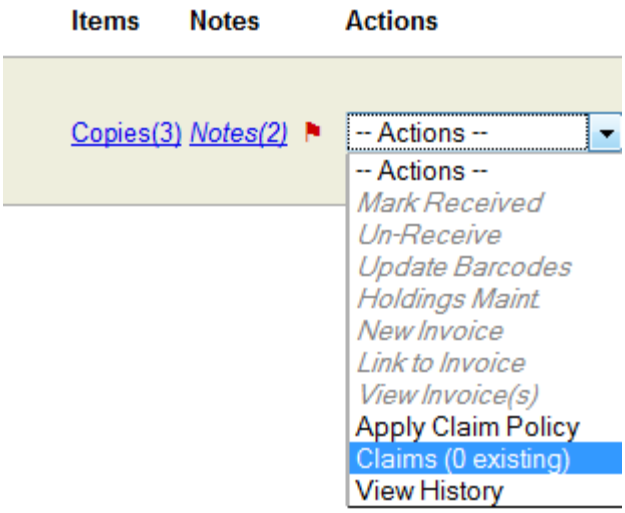
1. Open a selection list or purchase order.
2. Click the Actions drop down menu on the line item.
3. Click Change Claim Policy.
4. A drop down menu of claim policies will appear. Choose a claim policy to apply to the line item.
5. Click Save.

Claim an item

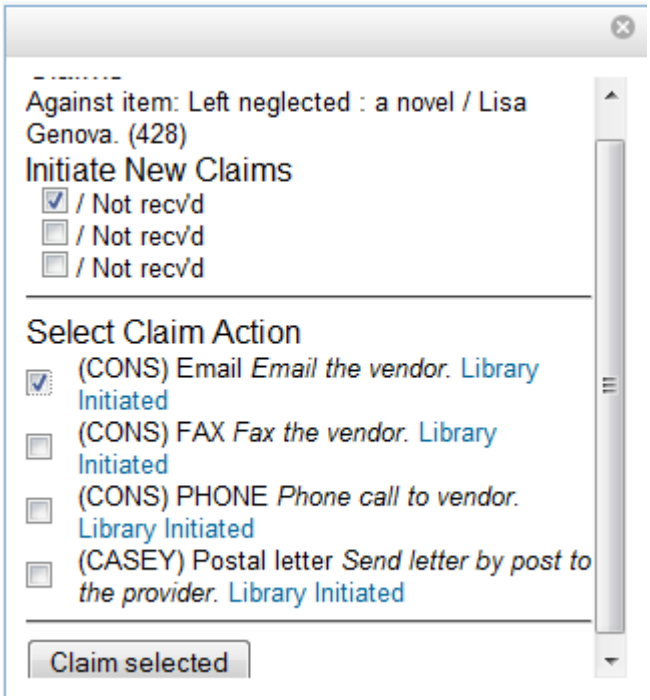
You can manually claim items at any time after the item has been ordered.

1. Open a purchase order.
2. Click the Actions drop down menu on the line item.

3. Click Claims. The number of existing claims appears in parentheses.



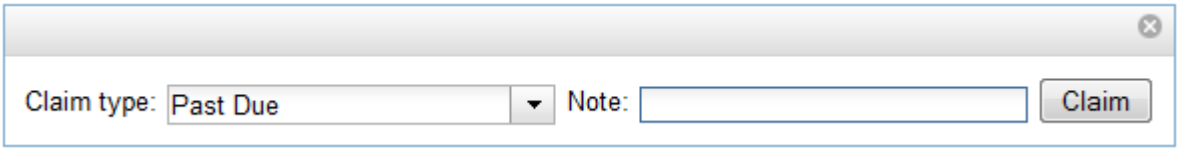
4. A drop down menu of items to be claimed and possible claim actions appears. Check the boxes adjacent to the item that you want to claim and the action that you will take. You can claim items that have not been received or cancelled.



5. Click Claim Selected.

6. Select a claim type from the drop down menu. Entering a note is optional.

7. Click Claim.



8. The number of existing claims on the line item updates, and a claim voucher appears. The voucher can be printed and mailed to the vendor to initiate the claim.

Produce a list of claim-ready items

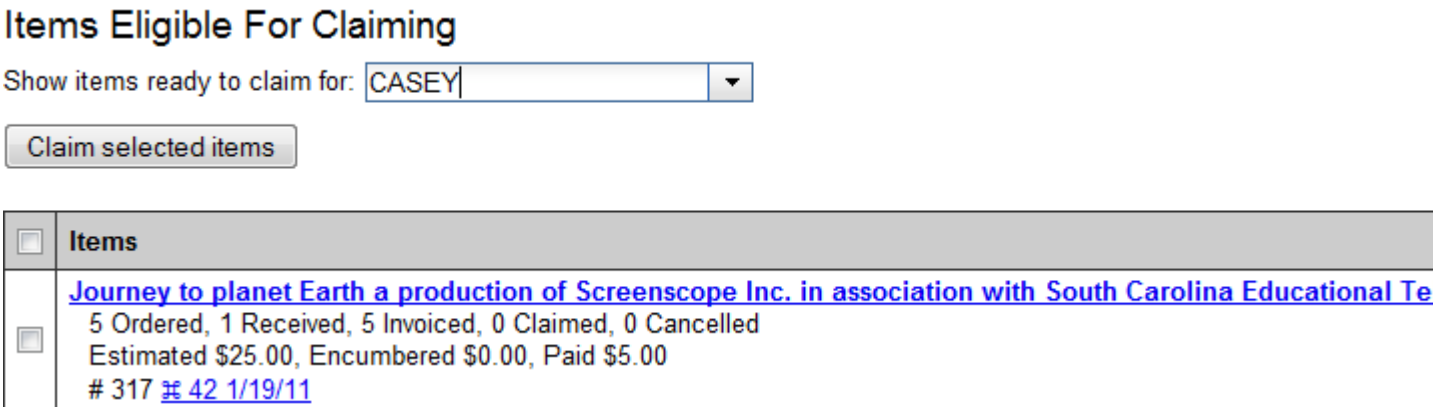
If an item has not been received and meets the conditions for claiming according to the item's claim policy, then the item will be eligible for claiming. Evergreen can produce a list of items, by ordering branch, which is ready to be claimed. You can use this list to manually claim items from your provider.

1. Click Acquisitions → Claim-Ready Items.

2. Choose a branch from the drop down menu to claim items that were ordered by this branch.

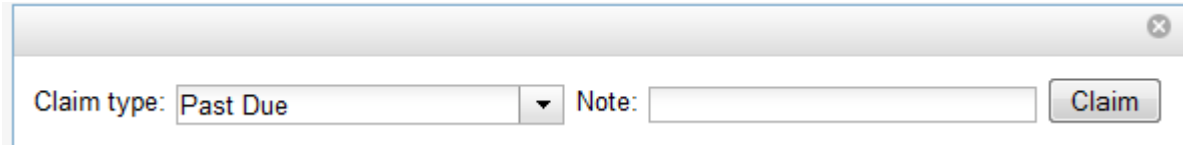
3. Any items that meet the conditions for claiming will appear.

4. Check the box adjacent to the line items that you wish to claim. Click Claim selected items.



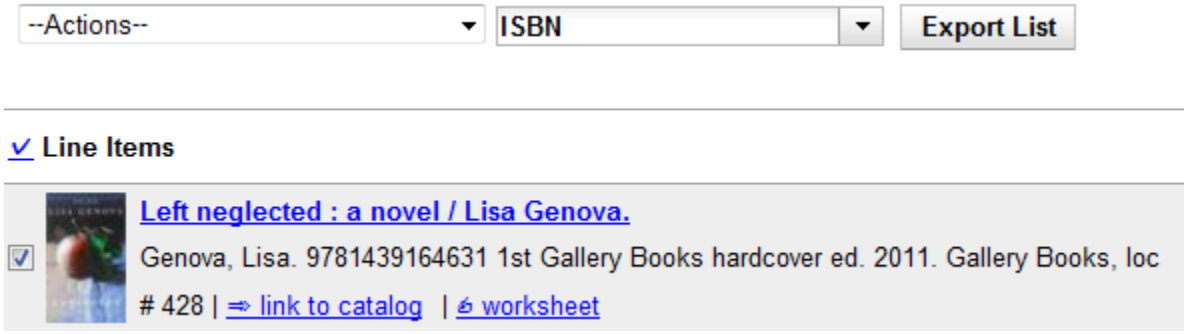
5. Select a claim type from the drop down menu. Entering a note is optional.

6. Click Claim.



Export Single Attribute List

You can export ISBNs, ISSNs, or UPCs as a file from the list of line item(s). A list of ISBNs, for example, could be uploaded to vendor websites when placing orders.



1. From a selection list or purchase order, check the boxes of the line items with attributes that you wish to export.
2. Click Actions → Export Single Attribute List.
3. Choose the line item attribute that you would like to export from the drop down list of attributes.
4. Click Export List.
5. Save the file to your computer.
6. Open the file. Choose a program (e.g. Excel) to open the file.

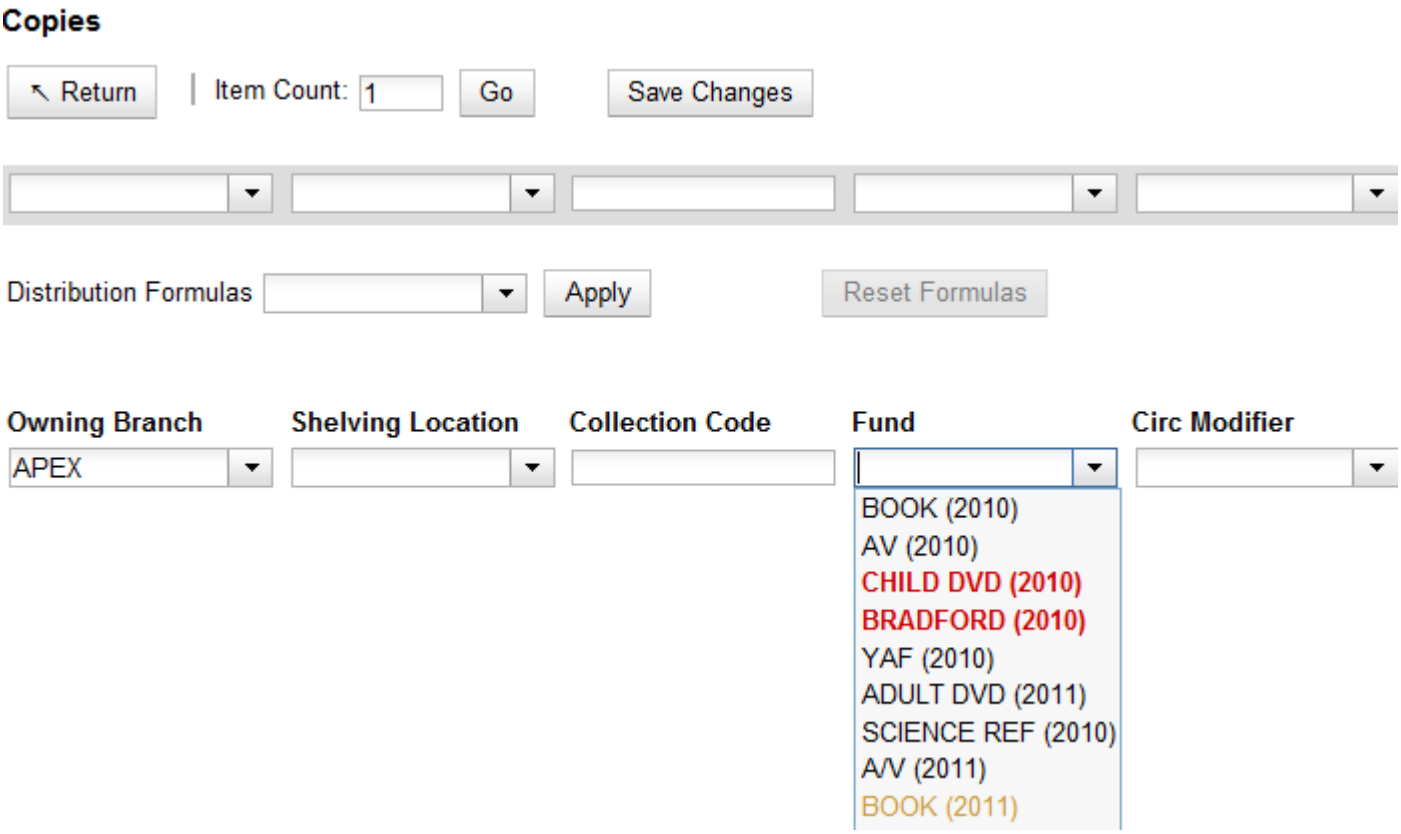
Funds

You can apply a single fund or multiple funds to copies on a selection list or purchase order. You can change the fund that has been applied to an item at any time on a selection list. You can change the fund that has been applied to an item on a purchase order if the purchase order has not yet been activated.

Funds can be applied to items from the Copies link that is located on a line item. Funds can also be applied to copies by batch updating line items and their attendant copies.

Apply funds to individual copies

1. Click the Copies link on the line item.
2. To apply a fund to an individual item, click the drop down arrow in the Fund field.



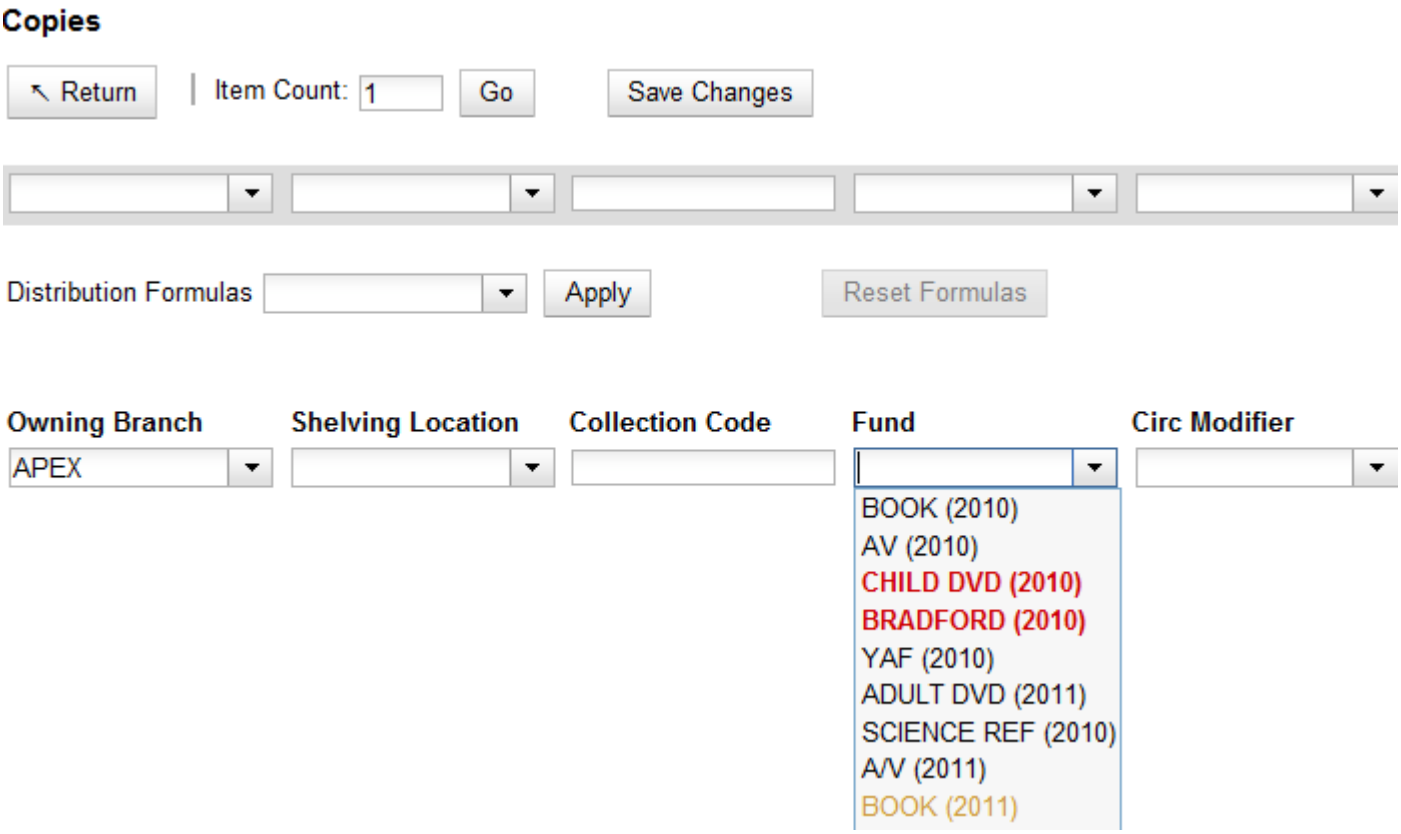
A yellow fund name indicates that the balance in the fund has dropped to the warning percent that was entered in the admin module. A red fund name indicates that the balance in the fund has dropped to the stop percent that was entered in the admin module. Funds that have been closed out will no longer appear on the drop down list.

3. To apply a fund to multiple items, see the section on line items for more information.

Apply funds to copies via batch updates to line items

You can apply funds to all copies on a line item(s) from the Actions menu on the selection list or the purchase order.

1. Check the boxes of the line items with copies to which you would like to apply funds.
2. Click Actions → Apply Funds to Selected Items.
3. Select the fund that you wish to apply to the copies.
4. Click Submit.



Invoice acquisitions

You can create invoices for purchase orders, individual line items, and blanket purchases. You can also link existing invoices to purchase order.

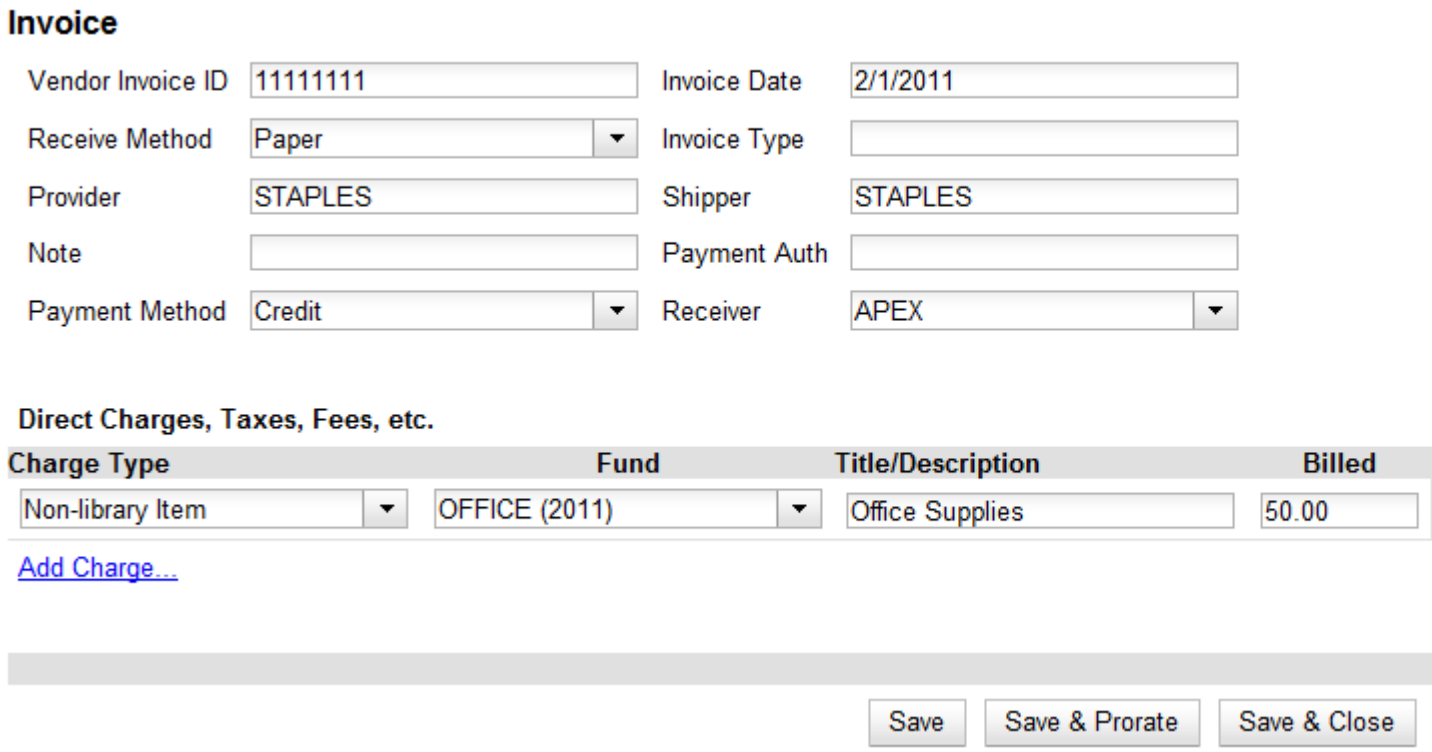
You can invoice items before you receive the items if desired. You can also reopen closed invoices, and you can print all invoices.

Create a blanket invoice

You can create a blanket invoice for purchases that are not attached to a purchase order.

1. Click Acquisitions → Create invoice.
2. Enter the invoice information in the top half of the screen.
3. To add charges for materials not attached to a purchase order, click Add Charge… This functionality may also be used to add shipping, tax, and other fees.
4. Select a charge type from the drop down menu.
5. Select a fund from the drop down menu.
6. Enter a Title/Description of the resource.

7. Enter the amount that you were billed.
8. Enter the amount that you paid.
9. Save the invoice.



Create an invoice for a line item

See the section on creating new invoices for line items for details.

Create an invoice for a purchase order

You can create an invoice for all of the line items on a purchase order. With the exception of fields with drop down menus, no limitations on the data that you enter exist.

1. Open a purchase order.
2. Click Create Invoice.
3. Enter a Vendor Invoice ID. This number may be listed on the paper invoice sent from your vendor.
4. Choose a Receive Method from the drop down menu.
5. The Provider is generated from the purchase order and is entered by default.
6. Enter a note (optional).
7. Select a payment method from the drop down menu.

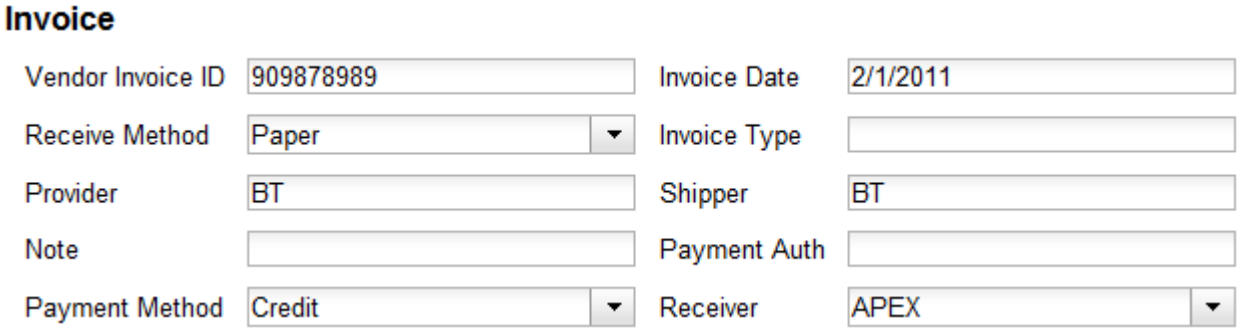
8. The Invoice Date is entered by default as the date that you create the invoice. You can change the date by clicking in the field. A calendar drops down.

9. Enter an Invoice Type (optional).

10. The Shipper defaults to the provider that was entered in the purchase order.

11. Enter a Payment Authorization (optional).

12. The Receiver defaults to the branch at which your workstation is registered. You can change the receiver by selecting an org unit from the drop down menu.



The bibliographic line items are listed in the next section of the invoice. Along with the title and author of the line items is a summary of copies ordered, received, invoiced, claimed, and cancelled. You can also view the amounts estimated, encumbered, and paid for each line item. Finally, each line item has a line item ID and links to the selection list (if used) and the purchase order.

13. Enter the number of items that were invoiced, the amount that the organization was billed, and the amount that the organization paid.

14. You have the option to add charge types if applicable. Charge types are additional charges that can be selected from the drop down menu. Common charge types include taxes and handling fees.

15. You have three options for saving an invoice. You can click Save, which saves the changes that you have made, but keeps the invoice open. You can click Save and Prorate, which enables you to save the invoice and prorate any additional charges, such as taxes, across funds, if multiple funds have been used to pay the invoice. You also can click Save and Close. Choose this option when you have completed the invoice.

You can re-open a closed invoice by clicking the link, Re-open invoice. This link appears at the bottom of a closed invoice.

Link an existing invoice to a purchase order

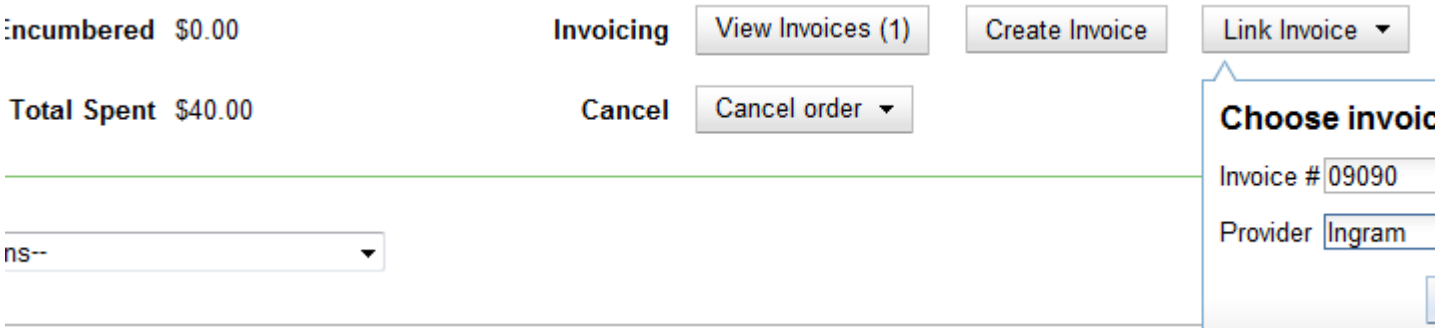
You can use the link invoice feature to link an existing invoice to a purchase order. For example, an invoice is received for a shipment with items on purchase order #1 and purchase order #2. When the invoice arrives, purchase order #1 is retrieved, and the invoice is created. To receive the items on purchase order #2, simply link the invoice to the purchase order. You do not need to recreate it.

1. Open a purchase order.

2. Click Link Invoice.

3. Enter the Invoice # and the Provider of the invoice to which you wish to link.

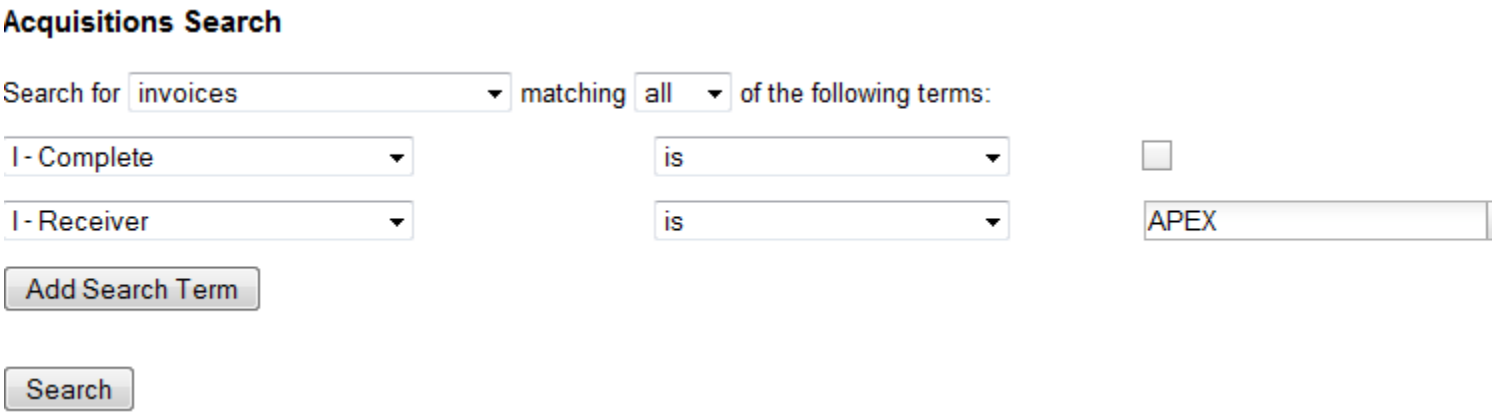
4. Click Link.



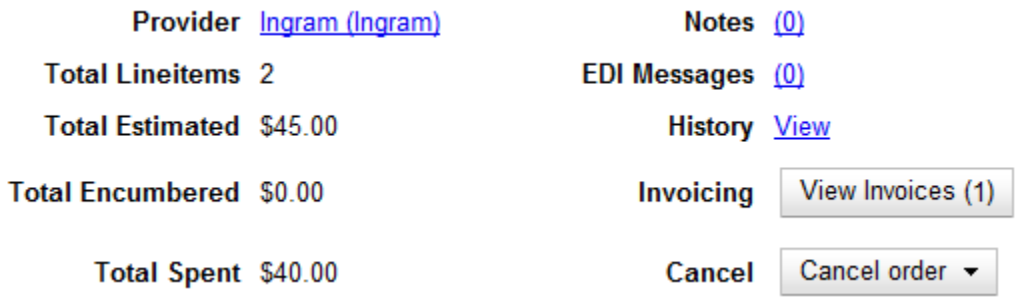
View an invoice

You can view an invoice in one of four ways: view open invoices; view invoices on a purchase order; view invoices by searching specific invoice fields; view invoices attached to a line item.

To view open invoices, click Acquisitions → Open invoices. This opens the Acquisitions Search screen. The default fields search for open invoices. Click Search.



To view invoices on a purchase order, open a purchase order, and click the View Invoices link. The number in parentheses indicates the number of invoices that are attached to the purchase order.



To view invoices by searching specific invoice fields, see the section on searching the acquisitions module.

To view invoices for a line item, see the section on line item invoices.

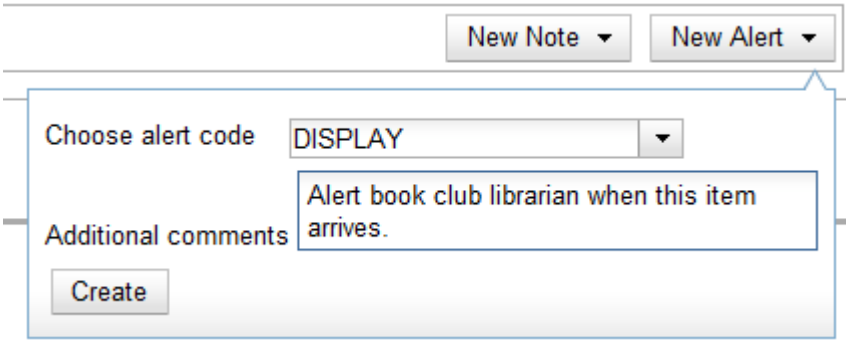
Line Items

Line items represent bibliographic records on a selection list or purchase order. One line item corresponds to one bibliographic record. Line items contain attributes, which are characteristics of the bibliographic record, such as ISBNs or Title. Line items also contain copy information, price information, and notes and alerts.

Add alerts to a line item

Alerts are pop up messages that appear when an item is received. Alerts can be printed on the line item worksheet.

1. Click the Notes link on the line item.
2. Click the New Alert drop down button.
3. Choose an alert code from the drop down menu.
4. Add additional comments if desired.
5. Click Create. The alert will display on the screen.



6. Click Return to return to the line item. When you return to the line item, a flag will appear to indicate that an alert is on the line item.

Add copies to a line item

Use the Copies link to add copy information to a line item. You can add copies to line items on a selection list or a purchase order.

1. Click the Copies link on a line item.

2. Enter the number of items that you want to order in Item Count, and click Go. The number of items that you want to order will display below.

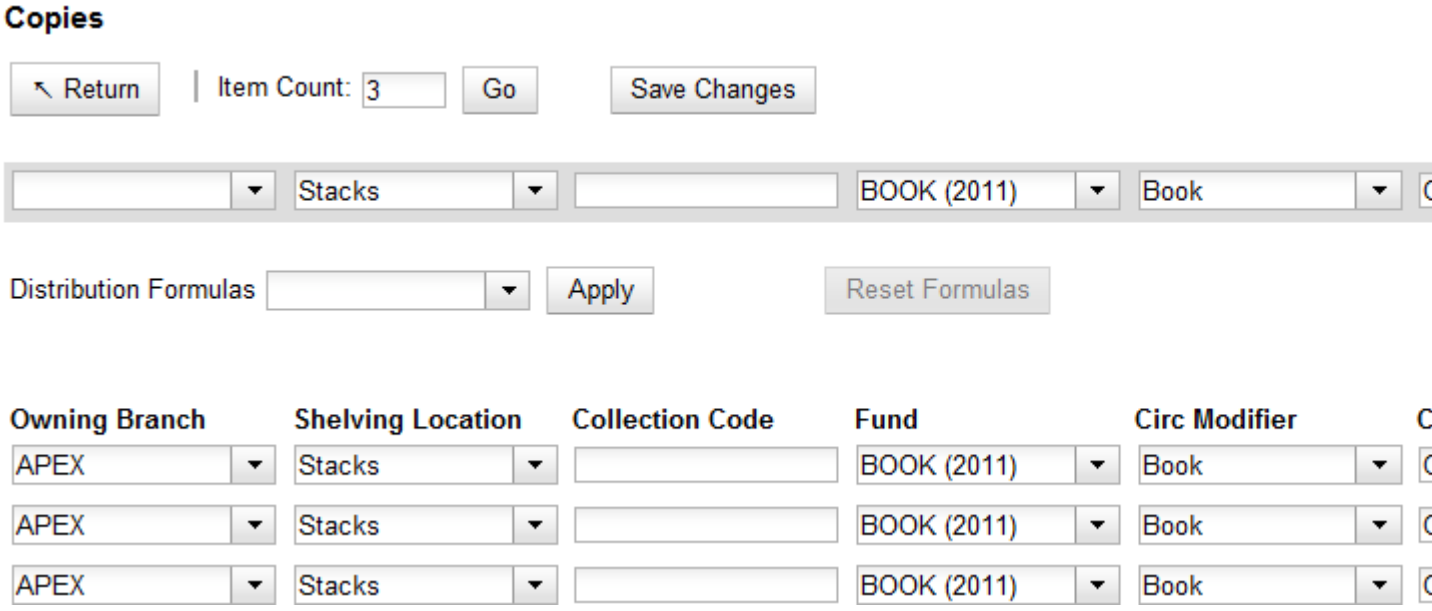
3. If desired, apply a Distribution Formula from the drop down list. Distribution formulas tell the ILS how many copies should be distributed to each location.

4. The owning branch and shelving location populate with entries from the distribution formula. Click Apply.

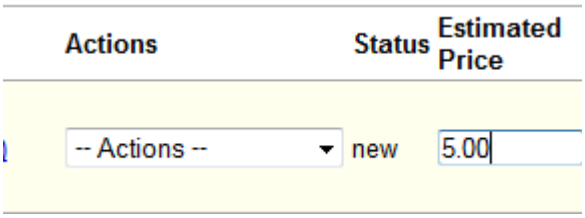
5. Look back at the top gray row of text boxes above the distribution formula. Each text box in this row corresponds to the columns below. Changes made here will be applied to all copies below. Click Batch Update.

6. Click Save Changes.

7. Click Return to return to the selection list or purchase order.



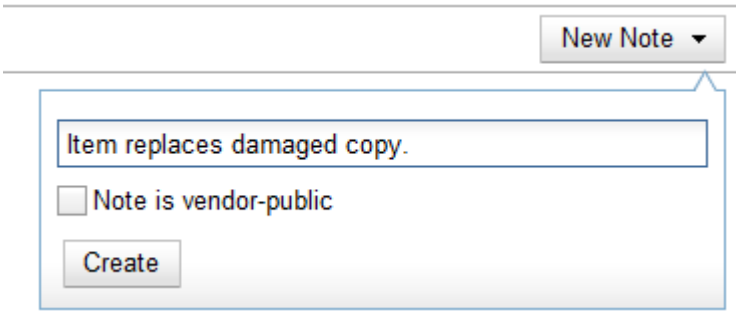
8. Add the item's price to the line item in the Estimated Price field.



Add notes to a line item

Notes on line items can include any additional information that you want to add to the line item. Notes can be internal or can be made available to providers. Notes appear in a pop up box when an item is received. Notes can be printed on line item worksheets, which can be printed and placed in books for processing.

1. Click the Notes link on the line item.
2. Click the New Note drop down button.
3. Enter a note.
4. You have the option to make this note available to your provider. Click the check box adjacent to Note is vendor-public.
5. Click Create. The note will appear on the screen.
6. Click Return to return to the line item. When you return to the line item, a number in parentheses adjacent to notes indicates how many notes are attached to the item.



Cancel a line item

For more information, see the section on cancelling/suspending acquistions.

Line item actions

Claims

See the section on claiming for more information.

Holdings maintenance

After an item has been received, click Actions → Holdings Maintenance to edit holdings. The Holdings Maintenance screen opens in a new tab.

Use the Link to invoice menu item to link the line item to an invoice that already exists in the ILS.

1. Click Actions → Link to Invoice.

2. A pop up box appears. Enter an invoice number.

3. Enter a provider. The field will auto-complete.

4. Click Link.

Mark received

See the section on receiving for more information.

New invoice

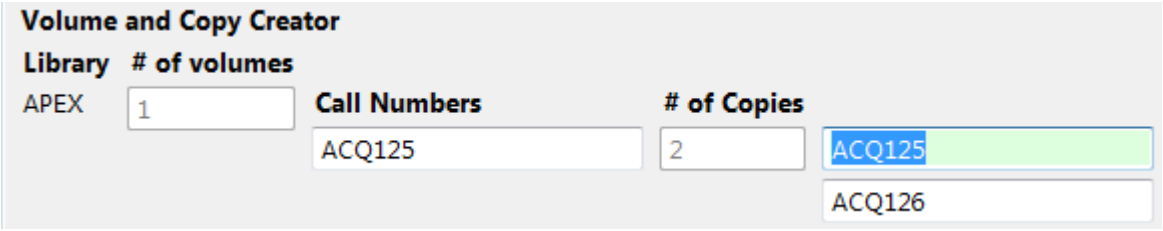
See the invoicing section for more information.

Un-receive

See the receiving/un-receiving section for more information.

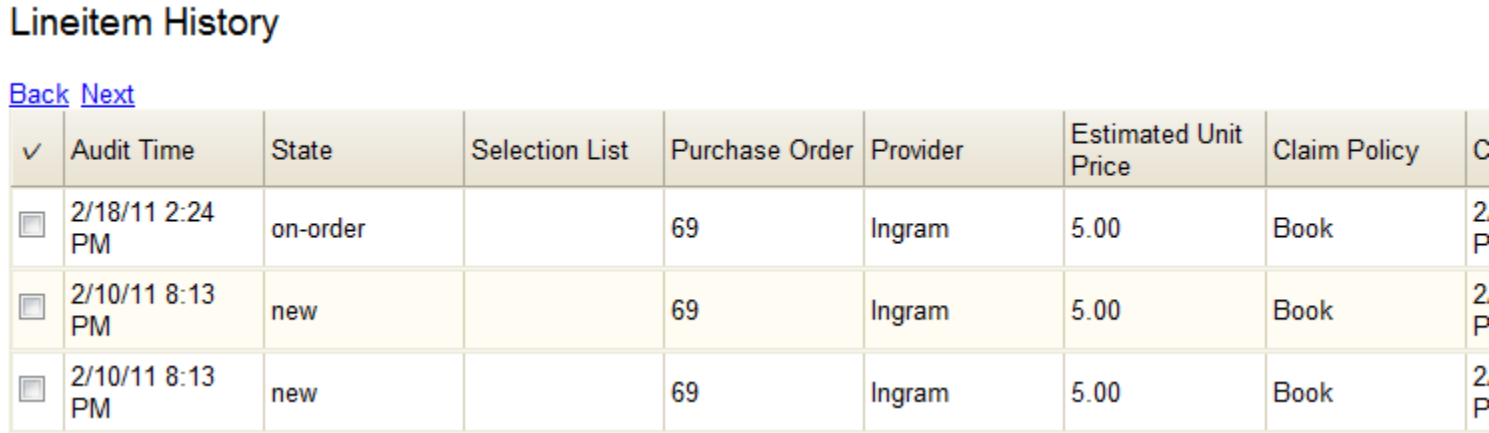
Update barcodes

After an item has been received, click Actions → Update Barcodes to edit holdings. The Volume and Copy Creator screen opens in a new tab.



View history

Click Actions → View history to view the changes that have occurred in the life of the line item.



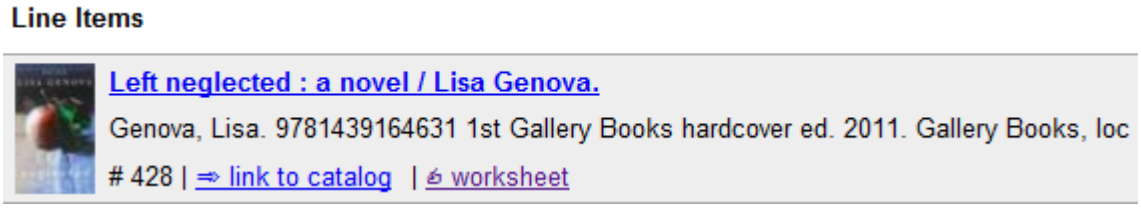
View invoice

Click Actions → View invoice to view any invoices that are attached to the line item.

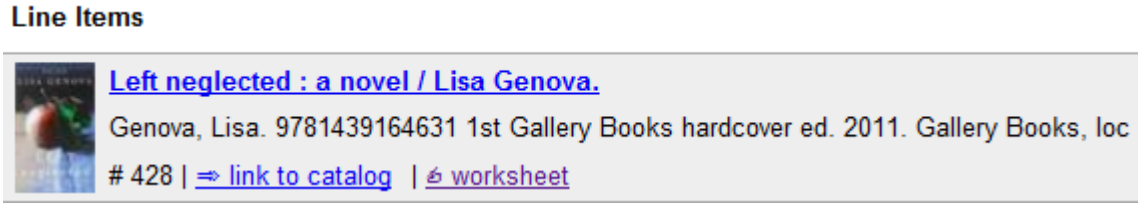
Line Item Worksheet

The Line Item Worksheet is a printable sheet that contains details about the line item, including alerts and notes, and distribution of the copies. This worksheet could be placed in a book that is sent to cataloging or processing.

1. From a selection list or purchase order, click the worksheet link on the line item.



2. The line item worksheet appears.



3. To print the worksheet, click the Print Page link in the top right corner.

Link line items to the catalog

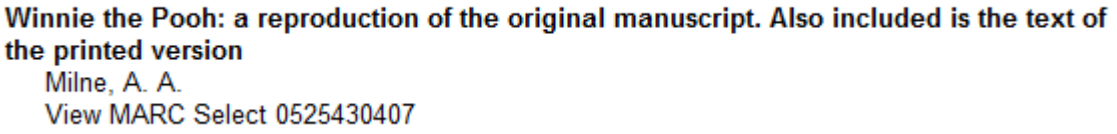
You can link a MARC record or brief record on a selection list to the corresponding MARC record in the catalog. This may be useful for librarians who have a brief MARC record in their catalog and want to import a better record that is attached to their selection list. No collision detection exists when importing an item into the selection list or catalog, so the link to catalog option enables you to search for a matching record and link to it from the selection list or purchase order. When you import the record from the purchase order, the record will overlay the linked record in the catalog.

1. From the line item, click Link to catalog.

2. In the text box that pops up, search terms, such as ISBN and title, are entered by default.

3. Click Search.

4. Result(s) appear. Click the link to View MARC, or Select the record to link it to the record on the selection list or purchase order.



5. The screen will reload, and the line item displays with a catalog link. The records are linked.

Load Bib Records and Items Into the Catalog

You can load bib records and items into the catalog at three different locations in the acquisitions module. You can import bib records and items (if holdings information is attached) when you upload MARC order records. Click Acquisitions → Load MARC Order Records and check the box adjacent to Load Bibs and Items into the ILS.You can import bib records and items into the catalog when you create a purchase order from a selection list. From the selection list, click Actions → Create Purchase Order. Check the box adjacent to Load Bibs and Items into the ILS to import the records into the catalog. You can import bib records and items into the catalog from a purchase order by clicking Actions → Load Bibs and Items. If you have not loaded bib records and items into the catalog before you activate a purchase order, then the ILS will automatically import the bib records and items into the catalog when you activate the purchase order.

Load Catalog Record IDs

The Load Catalog Record IDs function enables you to create line items from a list of catalog records whose record IDs are saved in a CSV file.

This would be useful if you want to batch order copies of items that your organization already owns. For example, after gathering a list of needed titles from your OPAC through a report, save the record IDs into a CSV file, upload the file into the ILS, and create a purchase order for the items.

1. Create a CSV file with the record ID of each catalog record in the first column of the spreadsheet. You can create this CSV file from a spreadsheet generated by a report, as suggested in the aforementioned example. You can also copy and paste record IDs from the catalog record into the CSV file.

Record IDs are auto-generated digits associated with each record. They are found in the Record Summary that appears at the top of each record.

2. Save the CSV file to your computer.
3. Click Acquisitions → Load Catalog Record IDs.
4. Click Load More Terms.
5. The screen will display the number of terms (record IDs) that have been loaded.
6. Click Retrieve Records. The records will appear as line items to which you can add copies, notes, and pricing information. Use the Actions menu to save these items to a selection list or purchase order.

Load MARC Order Records

The Load MARC Order Records screen enables you to upload MARC records that have been saved on your computer into the ILS. You can add the records to a selection list and/or to a purchase order. You can both create and activate purchase orders in one step from this interface. Also, from this interface, you can load bibs and items into the catalog.

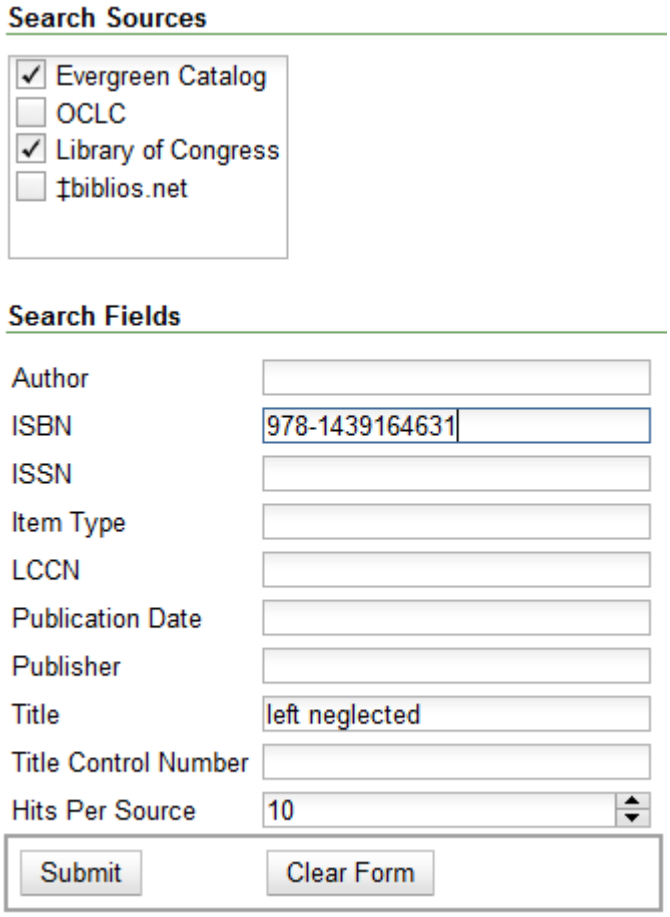
1. Click Acquisitions → Load MARC Order Records
2. If you want to upload the MARC records to a new purchase order, then click the check box adjacent to Create Purchase Order.
3. If you want to activate the purchase order at the time of creation, then click the check box adjacent to Activate Purchase Order.
4. If you want to load bibs and items into the catalog, then click the check box adjacent to Load Bibs and Items into the ILS.
5. Enter the name of the Provider. The text will auto-complete.
6. Select an org unit from the drop down menu. The context org unit is the org unit that `owns` the bib record. You should select a physical location rather than a political or administrative org unit as the context org unit. For example, the `Smith County Library System` is funding purchase of a copy of `Gone with the Wind`. The system owns the bib record, but it cannot receive the physical item. The acquisitions librarian will choose a physical branch of that system, a processing center or an individual branch, to receive the item.
7. If you want to upload the records to a selection list, you can select a list from the drop down menu, or type in the name of the selection list that you want to create.
8. Click Browse to search for the file of bibliographic records.
9. Click Upload.
10. A summary of the items that have been processed will appear.
11. Click the links that appear to view the purchase order or the selection list.

MARC Federated Search

The MARC Federated Search enables you to import bibliographic records into a selection list or purchase order from a Z39.50 source.

1. Click Acquisitions → MARC Federated Search.

2. Check the boxes of Z39.50 services that you want to search. Your local Evergreen Catalog is checked by default. Click Submit.

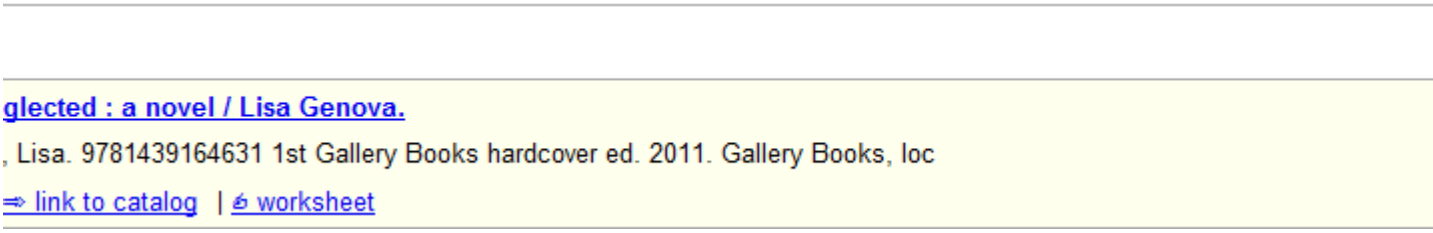


3. A list of results will appear. Click the Copies link to add copy information to the line item. See the section on Line Items for more information.

4. Click the Notes link to add notes or line item alerts to the line item. See the section on Line Items for more information.

5. Enter a price in the Estimated Price field.

6. You can save the line item(s) to a selection list by checking the box on the line item and clicking Actions → Save Items to Selection List. You can also create a purchase order from the line item(s) by checking the box on the line item and clicking Actions → Create Purchase Order.



Patron Requests

Purchase Orders

You can create a purchase order from a selection list, a batch upload of MARC order records, the View/Place Orders link in the catalog, or results from a MARC Federated Search. You can also create blanket purchase orders to which you can add brief records or generic charges and fees.

Activate a purchase order

Before you can active a purchase order, the following criteria must be met:

- The field, Activate Order?, is located in the top half of the purchase order. The answer adjacent to this field must be “Yes”.
- Each line item must contain an estimated price. If the Activate Order? field in the top half of the purchase order reads, “No: The lineitem has no price (ACQ_LINEITEM_NO_PRICE)”, then simply enter a price in the estimated price field, tab out of the field, and click Reload.

When the above criteria have been met, proceed with the following:

Look at the Activate Order? field in the top half of the purchase order. Click the hyperlinked Activate Order. When you activate the order, the bibliographic records and copies will be imported into the catalog, and the funds associated with the purchases will be encumbered.

Add brief records to a purchase order

To add brief records to a purchase order, see the section on adding brief records for more information. You can add brief records to new or existing purchase orders.

Add charges, taxes, fees, or discounts to a purchase order

You can add charges, taxes, fees, or discounts to a purchase order. These additional charges will be reflected in the amounts that are estimated and encumbered on the purchase order.

1. Open or create a purchase order.
2. Click New charge.
3. Select a charge type from the drop down menu.

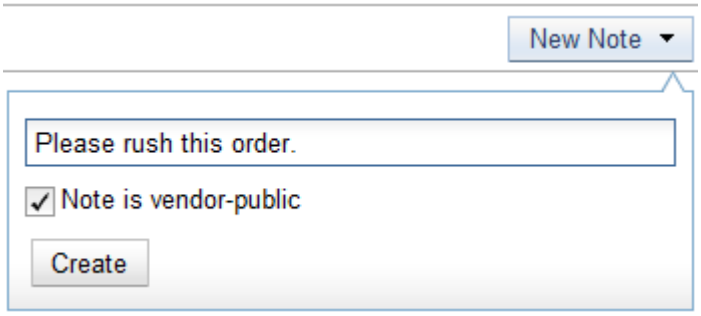
4. Select a fund from the drop down menu.
5. Enter a Title/Description, Author, and Note if applicable.
6. Enter an estimated cost.
7. Add another new charge, or click Save New Charges.

Discounts are not consistently supported in the 2.0 release.

Add notes to a purchase order

You can add notes to each purchase order. These can be viewed by staff and/or by the provider. By default, notes are only visible to staff.

1. Open a purchase order.
2. In the top half of the purchase order, you see a Notes field. The number of notes that are attached to the purchase order is hyperlinked in parentheses next to the Notes field.
3. Click the hyperlinked number.
4. Click New Note.
5. Enter the note. If you wish to make it available to the provider, click the check box adjacent to Note is vendor-public.
6. Click Create.

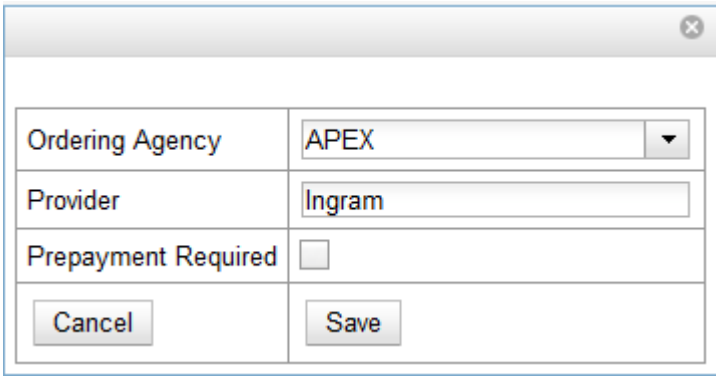


Cancel/Suspend a purchase order

To cancel or suspend a purchase order, see the cancel/suspend acquisitions section.

Create a purchase order

1. Click Acquisitions → Create Purchase Order.
2. A pop-up box appears. Select an owning library from the drop down menu.
3. Enter a provider in the box. The text will auto complete.
4. As necessary, check the box adjacent to Prepayment Required.
5. Click Save.



6. The purchase order has been created. You can now create a new charge type or add a brief record.

The Total Estimated is the sum of the prices. The Total Encumbered is the total estimated that is encumbered when the purchase order is activated. The Total Spent column automatically updates when the items are invoiced.

Mark ready for order

After an item has been added to a selection list or purchase order, you can mark it ready for order. This step is optional but may be useful to individual workflows.

1. If you want to mark part of a selection list ready for selector, then you can check the box(es) of the line item(s) that you wish to mark ready for selector. If you want to mark the entire list ready for selector, then skip to step 2.
2. ClickActions → Mark Ready for Order.
3. A pop up box will appear. Choose to mark the selected line items or all line items.
4. Click Go.
5. The screen will refresh. The line item will be highlighted gray, and the status will change to order-ready.

Name a purchase order

A new purchase order is given the purchase order ID as a default name. However, you can change that name to any grouping of letters or numbers. You can reuse purchase order names as long as a name is never used twice in the same year.

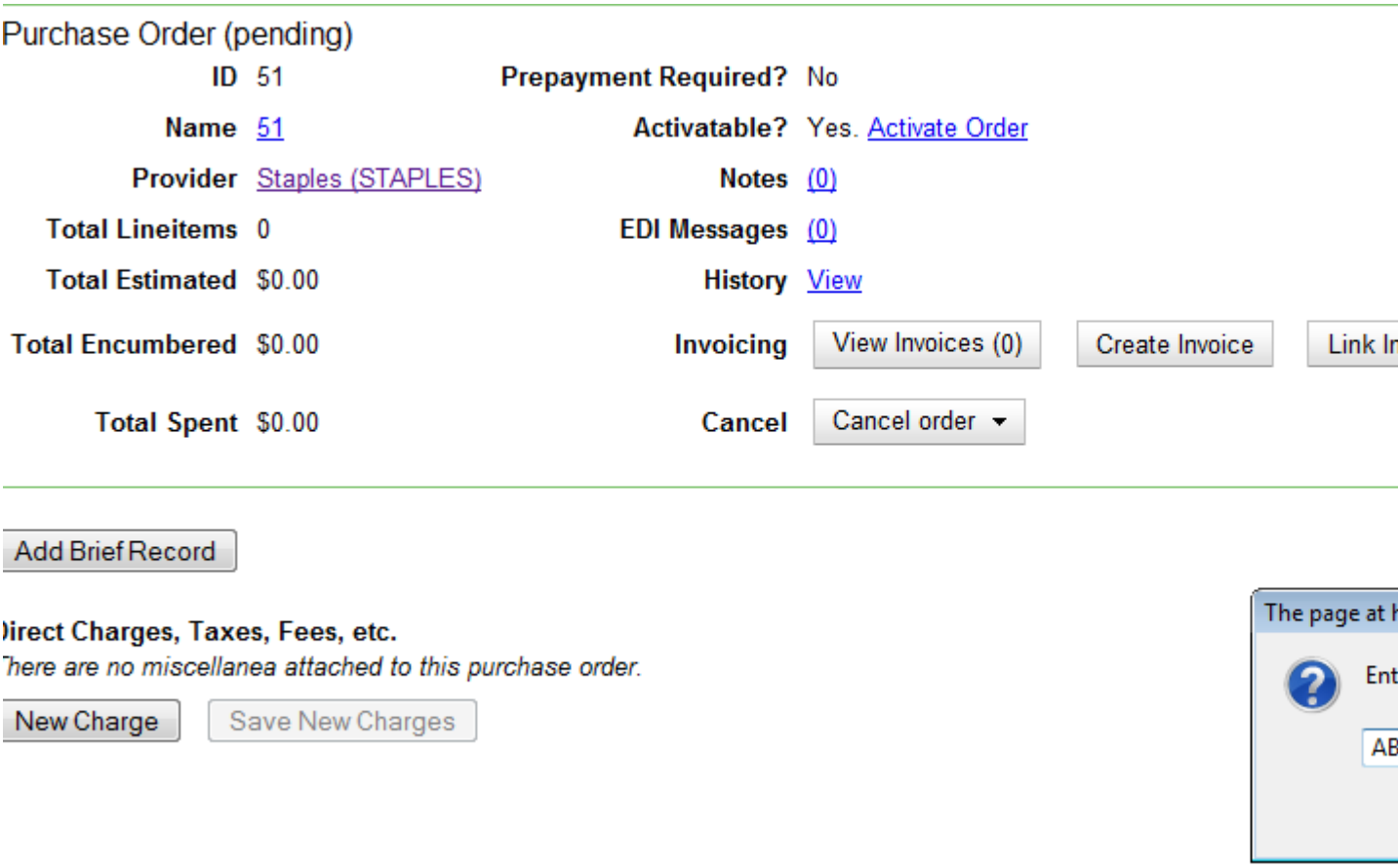
1. Open or create a purchase order.

2. The Name of the purchase order is in the top left column of the purchase order. The hyperlinked number is an internal ID number that Evergreen has assigned.

3. To change this number, click on the hyperlinked ID.

4. Enter a new purchase order number in the pop up box.

5. Click OK.



Print purchase orders

You can print a purchase order from the purchase order screen. If you add a note to a line item, the note will only appear in the Notes column on the printed purchase order if you make the note vendor-public. Currently, no notes appear in the Notes to the Vendor section of the printed purchase order.

1. Open a purchase order.

2. Click Actions → Print Purchase Order.

Purchase Order 86

date **20110218**

Vendor	Baker and Taylor	Ship to / Bill to	Apex Branch 123 Main St. Anywhere GA US 30303

Notes to the Vendor

PO#	ISBN or Item #	Title	Quantity	Unit Price	Line Total
86	9780380017607	Love Story	1		0
				Subtotal	0

Total Line Item Count: 1

Receive a purchase order

See the section on receiving acquistions for more information on receiving a purchase order.

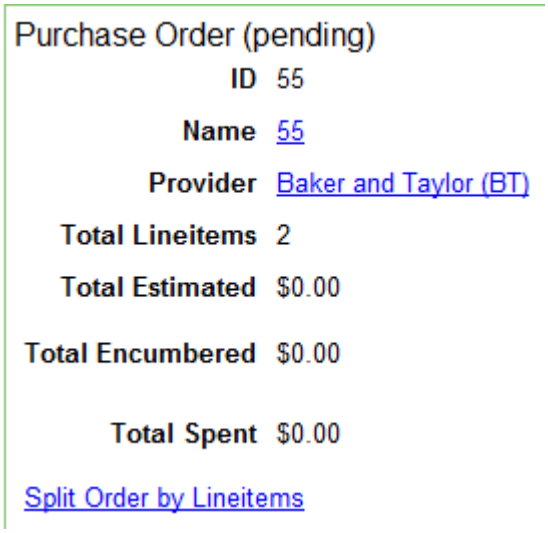
Split order by line items

You can create a purchase order with multiple line items, and then split the purchase order so that each line item is on separate purchase orders.

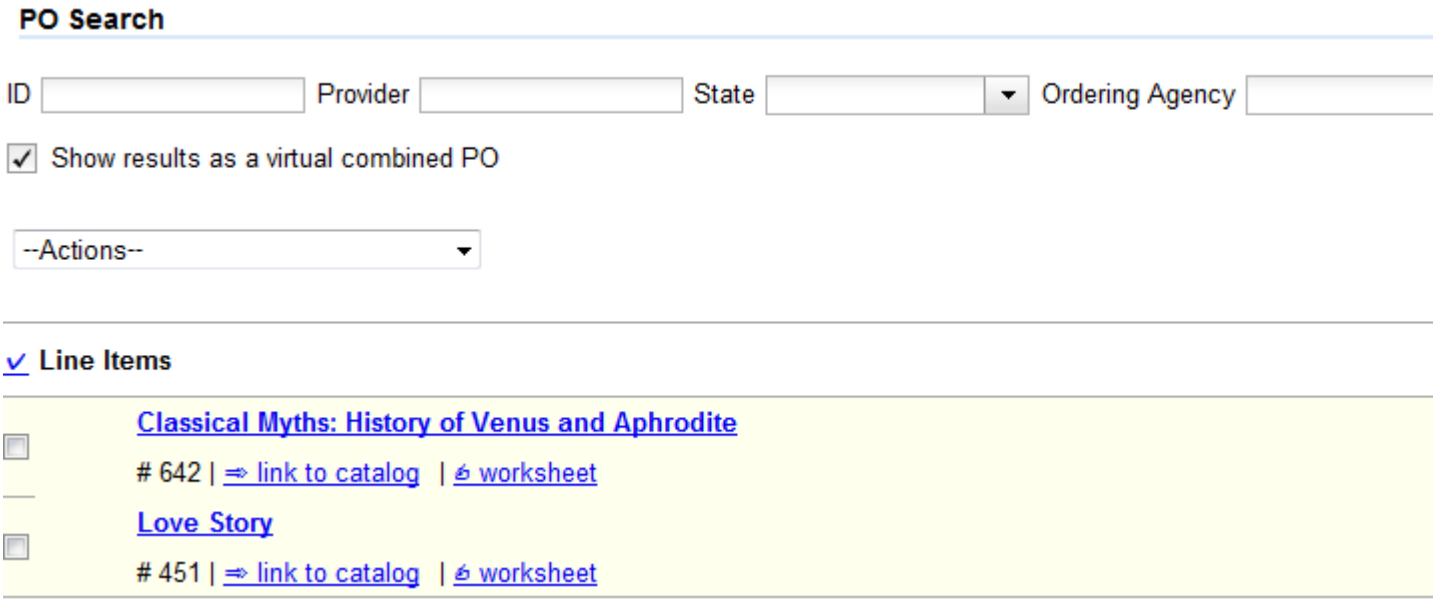
When a purchase order is in the status of pending, a link to split order by Lineitems appears in the bottom left corner of the top half of the screen.

1. Click Split Order by Lineitems.

2. A pop up box will confirm that you want to split the purchase order. Click OK to continue.



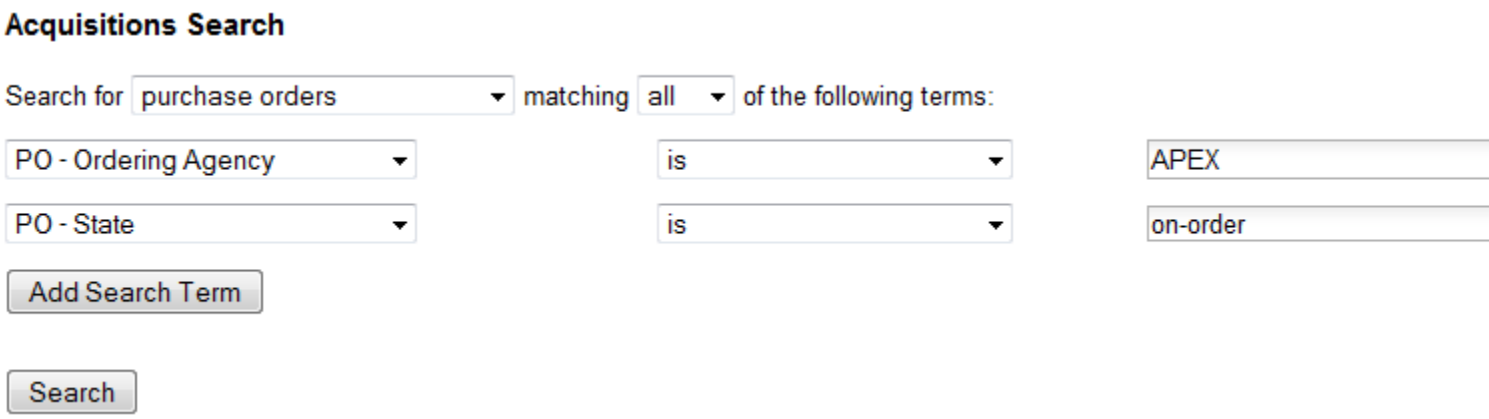
3. The items will display by default as a virtual combined purchase order. Future enhancements will allow you to activate the purchase order for each item from this screen.



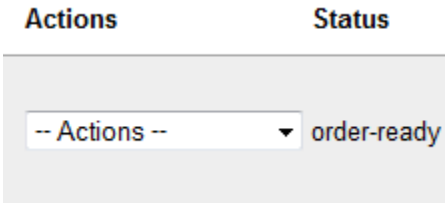
View On-Order Purchase Orders

You can view a list of on-order purchase orders by clicking Acquisitions → Purchase Orders. The ordering agency defaults to the branch at which your workstation is registered. The state of the purchase order defaults to on-order.

You can add more search terms by clicking Add Search Term. Search terms are “ANDed” together. Click Search to begin your search.



If you want to expand or change your search of purchase orders, you can choose other criteria from the drop down menus. See Searching Acquisitions for more information.



When searching by Org Unit, the exact ordering location must be selected. Searching for a consortium or system does not also display purchase orders or line items attached to child organizations.

View EDI messages on a purchase order

You can view electronic messages from your vendor about a specific purchase order.

1. Open a purchase order.
2. In the top half of the purchase order, you see an EDI Messages field. The number of messages that are attached to the purchase order is hyperlinked in parentheses next to the EDI Messages field.
3. Click the hyperlinked number to view the messages.

View Purchase Order History

In the top half of the purchase order, you can view the history of the purchase order. Click the View link in the History field.

Purchase Order History

Back Next

✓	Audit Time	Name	State	Ordering Agency	Provider	Create Time
☐	11/23/10 10:39 AM	13	on-order	1	Ingram	11/23/10 10:14 A
☐	11/23/10 10:39 AM	13	received	1	Ingram	11/23/10 10:14 A
☐	11/23/10 10:39 AM	13	received	1	Ingram	11/23/10 10:14 A
☐	11/23/10 10:39 AM	13	received	1	Ingram	11/23/10 10:14 A
☐	11/23/10 10:38 AM	13	on-order	1	Ingram	11/23/10 10:14 A
☐	11/23/10 10:38 AM	13	received	1	Ingram	11/23/10 10:14 A

Receiving

You can receive and un-receive entire purchase orders, line items, and individual copies. You can receive items before or after you invoice items.

Receive/un-receive copies

To receive copies, click the Copies link on the line item, and click the Mark Received link adjacent to each copy.

To un-receive copies, click the Copies link on the line item, and click the Un-Receive link adjacent to each copy.

Receive/un-receive line items

To receive a line item, click the Actions → Mark Received link on the line item.

To un-receive a line item, click the Actions → Un-receive link on the line item.

Receive/un-receive purchase orders

To receive a purchase order, click Actions → Mark Purchase Order as Received. The purchase order will have a state of received.

To un-receive a purchase order, click Actions → Un-Receive Purchase Order. The purchase will have a state of on order.

Searching

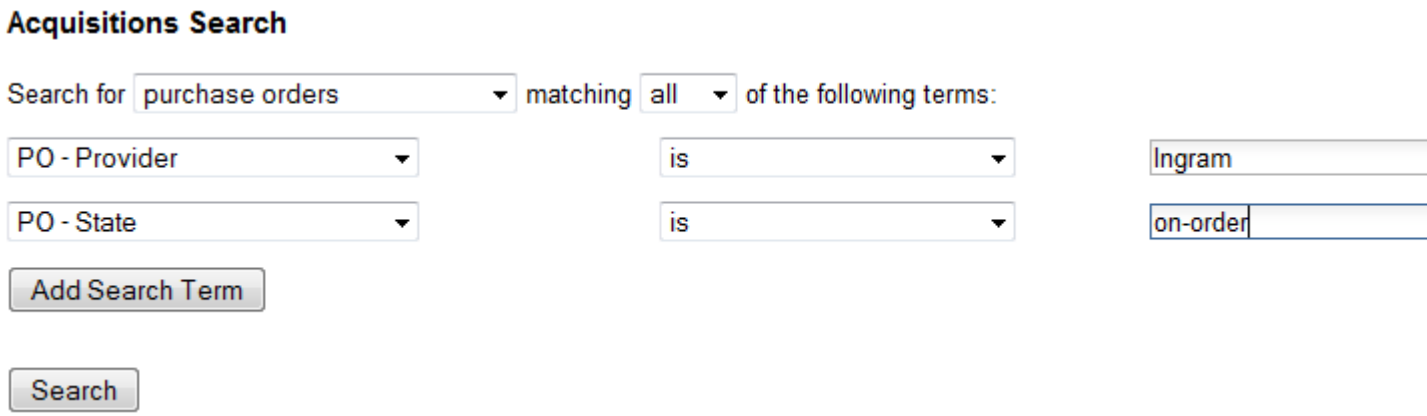
In the acquisitions module, you can search line items, line items and catalog records, selection lists, purchase orders, and invoices. To access the searching interface, click Acquisitions → General Search.

Users may wish to begin their acquisitions process by searching line items and catalog records. This ensures that they do not purchase an item that the library already owns or is on another selection list or purchase order.

1. Choose the object that you would like to search from the drop down menu.

2. Next, refine your search by choosing the specific fields that you would like to search. Click Add Search Term to add more fields. Search terms are “ANDed” together. Click the red X at the end of each row to delete search terms. Some search terms will be disabled depending on your choice of items to search.

3. After you have added search term(s), click Search or click the **Enter** key. A list of results appears.



4. If you want to edit your search, click the Reveal Search button in the top right corner of the results screen to display your search.

When searching by Org Unit, the exact ordering location must be selected. Searching for a consortium or system does not also display purchase orders or line items attached to child organizations.

Selection Lists

Selection lists allow you to create, manage, and save lists of items that you may want to purchase. To view your selection list, click Acquisitions → My Selection Lists. Use the general search to view selection lists created by other users.

Create a selection list

Selection lists can be created in four areas within the module. Selection lists can be created when you Add Brief Records, Upload MARC Order Records, or find records through the MARC Federated Search. In each of these interfaces, you will find the Add to Selection List field. Enter the name of the selection list that you want to create in that field.

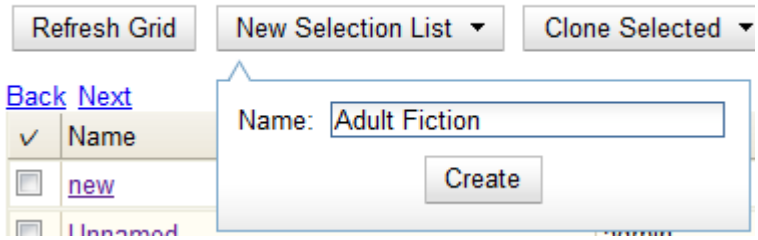
Selection lists can also be created through the My Selection Lists interface:

1. Click Acquisitions → My Selection Lists.

2. Click the New Selection List drop down arrow.

3. Enter the name of the selection list in the box that appears.

4. Click Create.



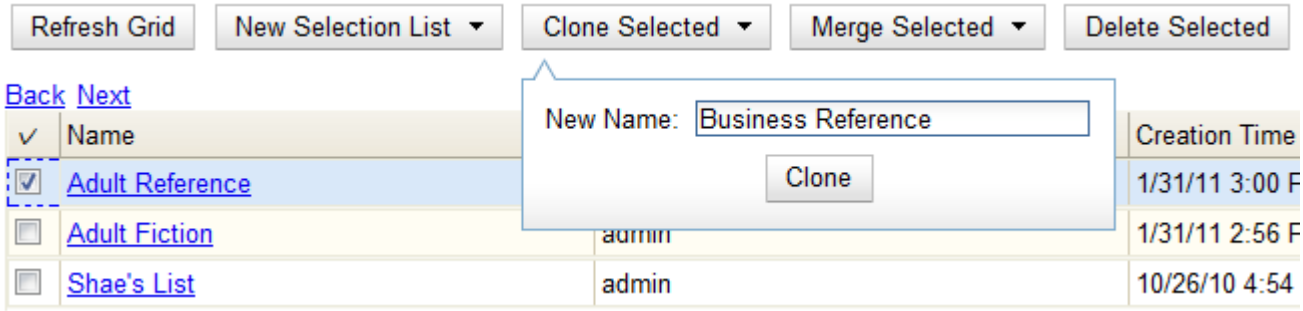
Add items to a selection list

You can add items to a selection list in one of three ways: add a brief record; upload MARC order records; add records through a federated search; or use the View/Place Orders menu item in the catalog.

Clone selection lists

Cloning selection lists enables you to copy one selection list into a new selection list. You can maintain both copies of the list, or you can delete the previous list.

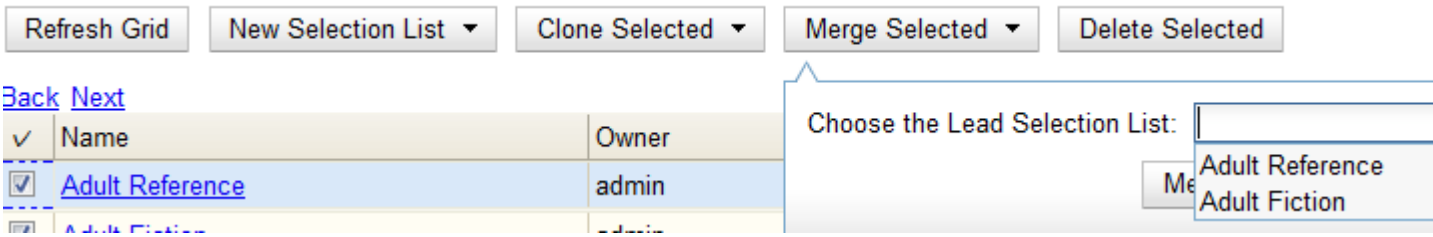
1. Click Acquisitions → My Selection Lists.
2. Check the box adjacent to the list that you want to clone.
3. Click Clone Selected.
4. Enter a name into the box that appears, and click Clone.



Merge selection lists

You can merge two or more selection lists into one selection list.

1. Click Acquisitions → My Selection Lists.
2. Check the boxes adjacent to the selection lists that you want to merge, and click Merge Selected.
3. Choose the Lead Selection List from the drop down menu. This is the list to which the items on the other list(s) will be transferred.



4. Click Merge.

Delete selection lists

You can delete selection lists that you do not want to save. You will not be able to retrieve these items through the General Search after you have deleted the list. You must delete all line items from a selection list before you can delete the list.

1. Click Acquisitions → My Selection Lists.
2. Check the box adjacent to the selection list(s) that you want to delete.
3. Click Delete Selected.

Mark Ready for Selector

After an item has been added to a selection list or purchase order, you can mark it ready for selector. This step is optional but may be useful to individual workflows.

1. If you want to mark part of a selection list ready for selector, then you can check the box(es) of the line item(s) that you wish to mark ready for selector. If you want to mark the entire list ready for selector, then skip to step 2.
2. Click Actions → Mark Ready for Selector.
3. A pop up box will appear. Choose to mark the selected line items or all line items.
4. Click Go.
5. The screen will refresh. The marked line item(s) will be highlighted pink, and the status changes to selector-ready.

Convert selection list to purchase order

Use the Actions menu to convert a selection list to a purchase order.

1. From a selection list, click Actions → Create Purchase Order.
2. A pop up box will appear.
3. Select the ordering agency from the drop down menu.
4. Enter the provider.
5. Check the box adjacent to prepayment required if prepayment is required.
6. Choose if you will add All Lineitems or Selected Lineitems to your purchase order.
7. Check the box if you want to Import Bibs and Create Copies in the catalog.
8. Click Submit.

View/Place Orders

1. Open a bib record.
2. Click Actions for this Record → View/Place Orders.
3. Click Add to Selection List, or click Create Purchase Order.
4. See the documentation on purchase orders and/or selection lists for instructions to proceed.

Chapter 6. Cataloging

Working with the MARC Editor

The MARC Editor allows MARC tags, sub-fields, and indicators to be edited.

OPAC icons for text, moving pictures and sound rely on correct MARC coding in the leader and the 008, as do OPAC search filters such as publication date, item type, or target audience. Bibliographic matching and de-duplicating also need correct and consistent MARC coding in particular tags.

Editing MARC Records

1. Retrieve the record.
2. Actions for this Record → MARC Edit .
3. The MARC record will display.
4. Select viewing and editing options, if desired.
   - Stack subfields to display each subfield on its own line.
   - Flat-Text Editor switches to a plain-text (menmonic) MARC format. The advantage of this format is being able to copy and paste multiple lines. It also allows the use of tools like MarcEdit (http://people.oregonstate.edu/~reeset/marcedit/html/index.php). Unclick the box to switch back.
   - Fast Item Add allows attaching items quickly with call number and barcode. When Save is clicked, the **INSERT CROSS-REFERENCE** Copy Editor will open.
5. Make changes as desired.
   - Right click into a tag field to add/remove rows or replace tags.
   - With the cursor at the end of a subfield, click **Control + D** (**Control + I** in Mac OSX) to insert a new subfield.
   - To remove a subfield, click **Shift + Delete**
   - To work with the data in a tag or indicator, click or **Tab** into the required field. *Right click* to view acceptable tags or indicators.
   -

The MARC Editor may be navigated using keyboard shortcuts. Click Help to see the shortcut menu from within the MARC Editor.

6. When finished, click Save Record . The record stays open in the editor. You can close the tab or switch to another view under Actions for this Record (for example to view it as it appears in the OPAC).

Overlaying MARC Records

Overlaying a MARC record replaces an existing MARC record while leaving all holdings, holds, active circulations, bills, and fines intact.

In Evergreen, a record must be *marked* for overlay. The mark for overlay is by login. Only one record at a time may be marked for overlay. When another record is marked for overlay, the previously marked item is *de-marked* . Once a record is marked, it remains marked until overlaid or until the user logs out of Evergreen.

Marking a record for overlay

1. Search for and retrieve a record for overlay.
2. Select Actions for this Record → Mark for Overlay . Record is now *marked* .

Overlaying the marked record

1. Once the record is marked for overlay, proceed to search for and import the new record from a Z39.50 target .
2. Click MARC Editor for Overlay . The TCN of the Evergreen record marked for overlay is displayed.
3. The record displays in MARC Edit view. Edit the record as necessary.
4. Click Overlay Record .
5. The existing record will display along with a prompt to confirm the overlay. Panes may be moved to view the record in entirety, if required.
6. Click Overlay .
7. Confirm the overlay. The record in Evergreen is overlaid with the new MARC record. All pre-existing holdings remain intact.

Validating MARC Records

Validation checks headings that are subject to authority control (subjects, names, uniform titles) by comparing them to the authority records that are available in the system.

1. Click Validate .
2. Scroll through the record looking for fields that have turned red.
3. Right-click on a field that is red. The closest valid headings will display.
4. To apply a heading, mouse over it and click Apply Selected .
5. Fields that do not validate are not necessarily invalid--there may not be an authority record for that name/subject/title. If you have the appropriate permissions, you can add an authority record based on the existing record.

Adding Holdings

Unified Volume Copy Creator

The cataloging module in Evergreen version 2.1 now includes a unified volume/copy creator that enables a user to create volumes and copies in a unified screen. This function consolidates the process of creating volume and copy records.

Administrative Settings

By default, the cataloging interface that existed in 2.0 will display in 2.1. To use the **Unified Volume/Copy Creator**, you must turn it on in the Admin module. You must log out of Evergreen and log back in for the changes to take effect.

To turn on the **Unified Volume/Copy Creator**:

1. Select **Admin # Local Administration # Library Settings Editor**
2. Scroll down to **GUI: Unified Volume/Item Creator/Editor**, and click **Edit**.
3. Select **True** to turn on the editor.

After you turn on the **Unified Volume/Copy Creator**, it will display, by default, in a vertical panel. You may display the **Unified Volume/Copy Creator** in a horizontal panel. You must log out of Evergreen and log back in for the changes to take effect.

To choose the horizontal display of the **Unified Volume/Copy Creator**:

1. Select **Admin # Local Administration # Library Settings Editor**.
2. Scroll down to **GUI: Horizontal layout for Volume/Copy Creator/Editor**, and click **Edit**.
3. Select **True** to view the panel horizontally.

Add volumes and items using the Unified Volume/Copy Creator

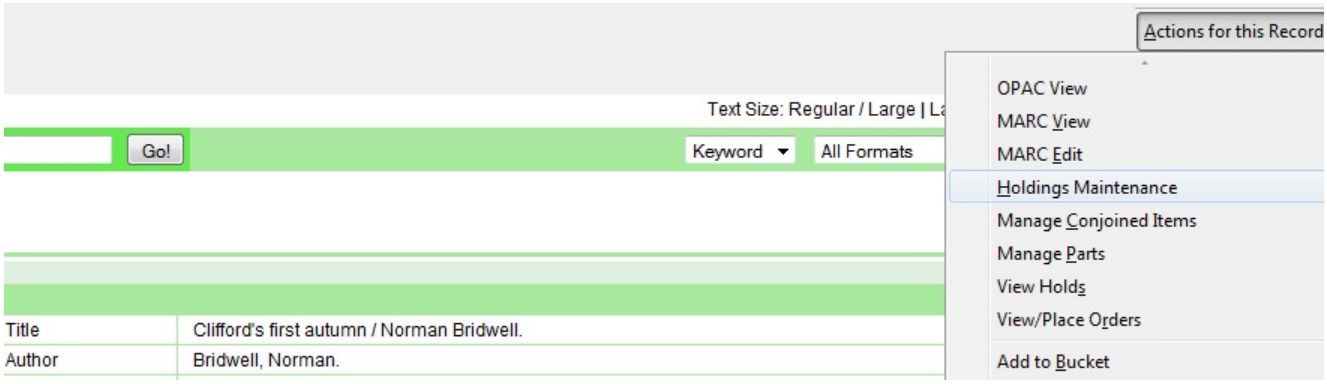
The **Unified Volume/Copy Creator** enables you to edit call numbers and copy information in separate panes on a single, unified screen.

The **Unified Volume/Copy Creator** appears when you access the following links:

- Holdings Maintenance # Add items
- Holdings Maintenance # Add volumes
- Holdings Maintenance # Edit items
- Holdings Maintenance # Replace barcode
- Create New MARC Record # Fast Item Add
- Record Summary # Add Volumes.

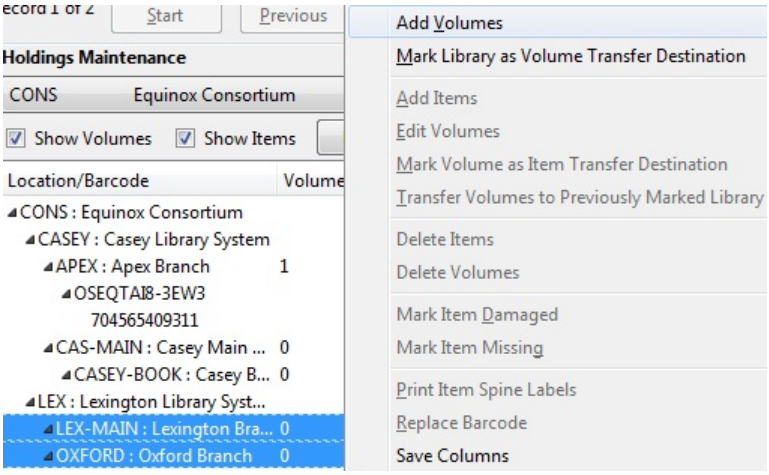
The ability to **Add Volumes** from the **Record Summary** is also a new functionality in 2.1.

In the following example, you will use the **Unified Volume/Copy Creator** to add multiple volumes and copies to bibliographic record from the Holdings Maintenance screen:



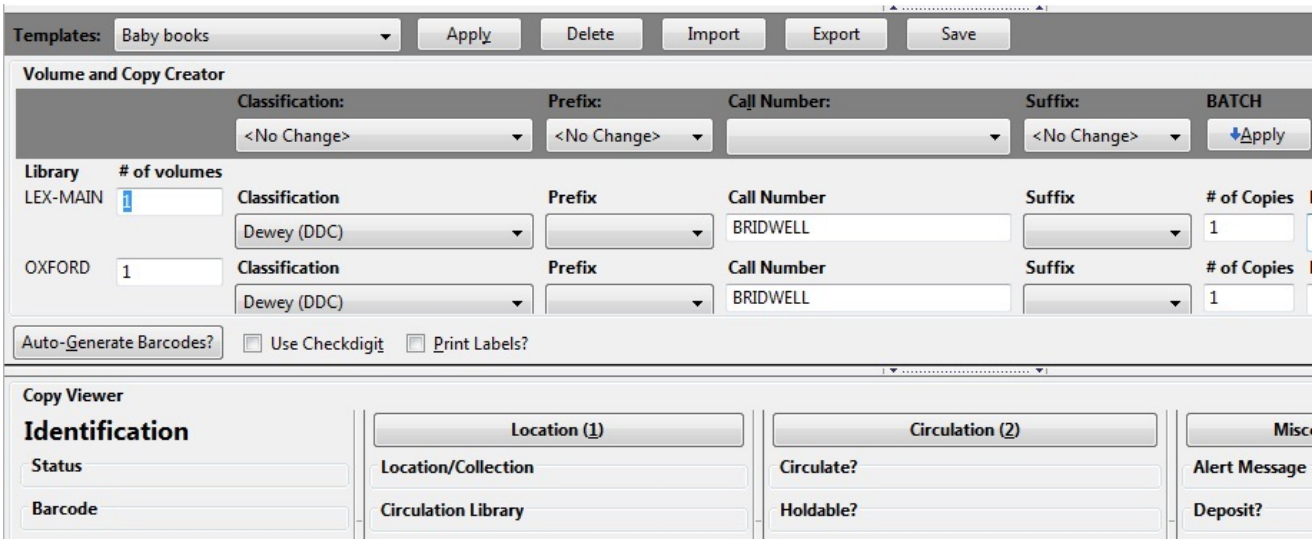
- The **Record Summary** appears in the top panel.
- The **Copy Templates** enables you to predefine copy settings. This panel displayed in the **Copy Editor** in 2.0.
- The **Volume and Copy Creator** enables you to add volumes and copies individually or in batch.
- The **Copy Viewer** enables you to set parameters for each copy.

1. Retrieve a record.
2. Click the **Actions for this Record -# Holdings Maintenance**
3. On the **Holdings Maintenance** screen, highlight the branch(es) to which you would like to add volumes.
4. Right click, and click **Add Volumes.**



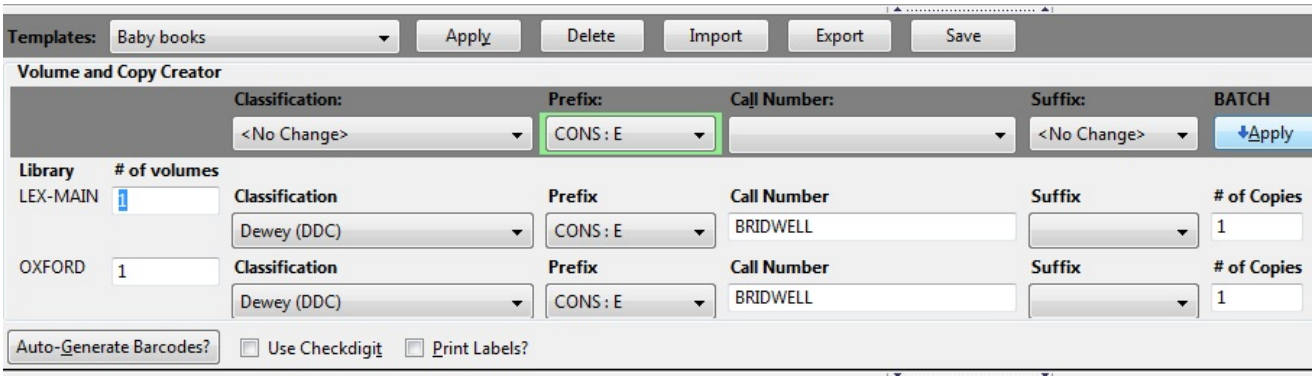
5. The **Unified Volume/Copy Creator** opens.
6. A **Classification** and a **Call Number** appear for each volume.

The **Classification** is derived from following setting in the Admin module: **Admin # Local Administration # Library Settings Editor # Cataloging: Default Classification Scheme**. The **Call Number** is derived from the MARC record.

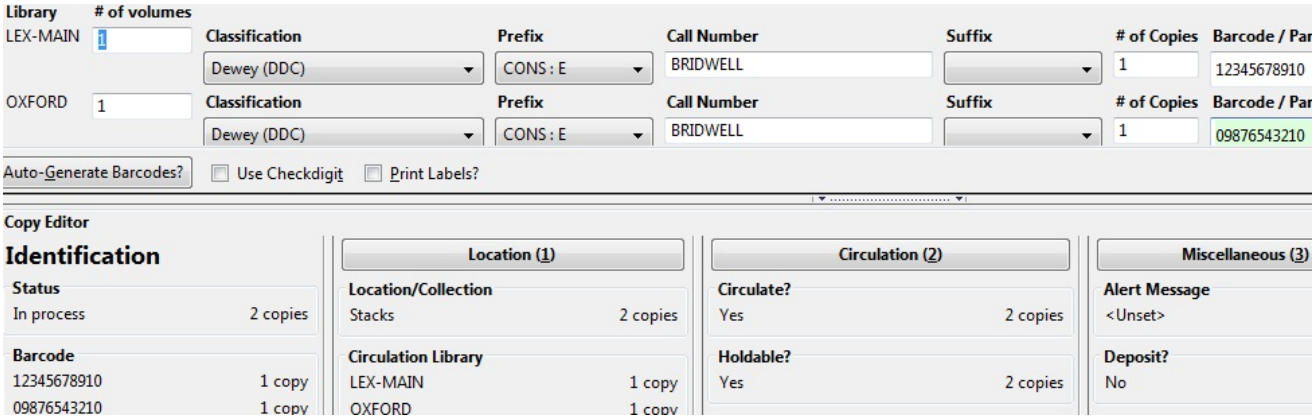


7. You can add volume and copy information to each row individually or in batch. In the dark gray row, select a classification, prefix, call number, and suffix (if needed) from the drop down menus and apply those settings in batch to all of the volumes that you created. Click **Apply**.

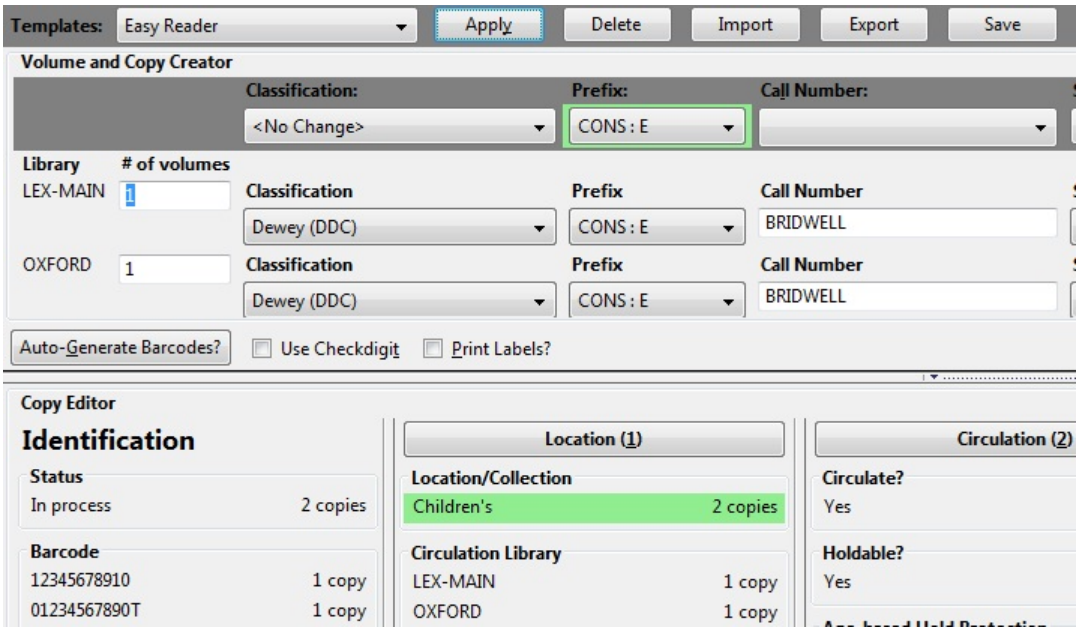
The prefix and suffix drop down menus are populated by entries in the **Admin** menu. See Call Number Prefixes and Suffixes.



8. Enter the number of copies and barcodes that you want to add. The barcodes that you enter will populate in the **Copy Editor**.



9. If desired, select a copy template from the template drop down menu, and click **Apply**. Changes to copies appear in green.



10. Make any other changes that you would like to make in the **Copy Editor**.

11. Click **Create Volumes/Items.**

12. The **Holdings Maintenance** screen will refresh to show the addition of the volumes and copies.

Monograph Parts

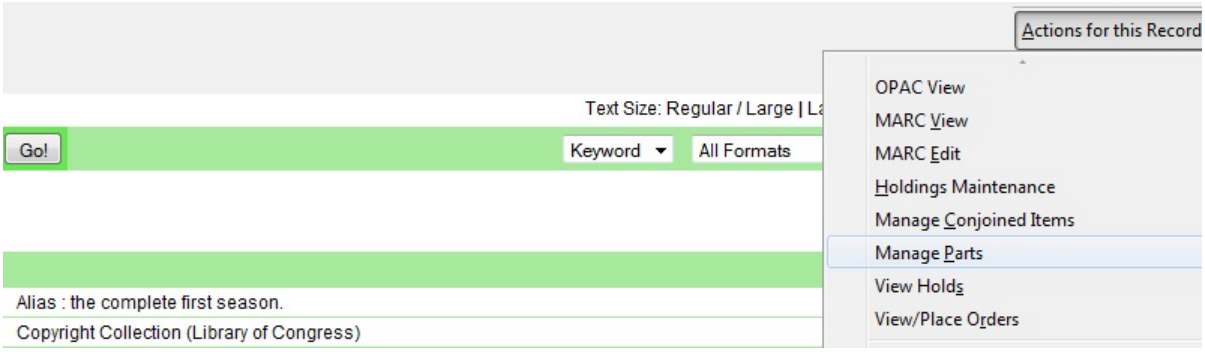
Monograph Parts enables you to differentiate between parts of monographs or other multi-part items. This feature enables catalogers to describe items more precisely by labeling the parts of an item. For example, catalogers might identify the parts of a monograph or the discs of a DVD set. This feature also allows patrons more flexibility when placing holds on multi-part items. A patron could place a hold on a specific disc of a DVD set if they want to access a specific season or episode rather than an entire series.

No new permissions or administrative settings are needed to use this feature.

To add a monograph part to an existing record in the catalog:

1. Retrieve a record.

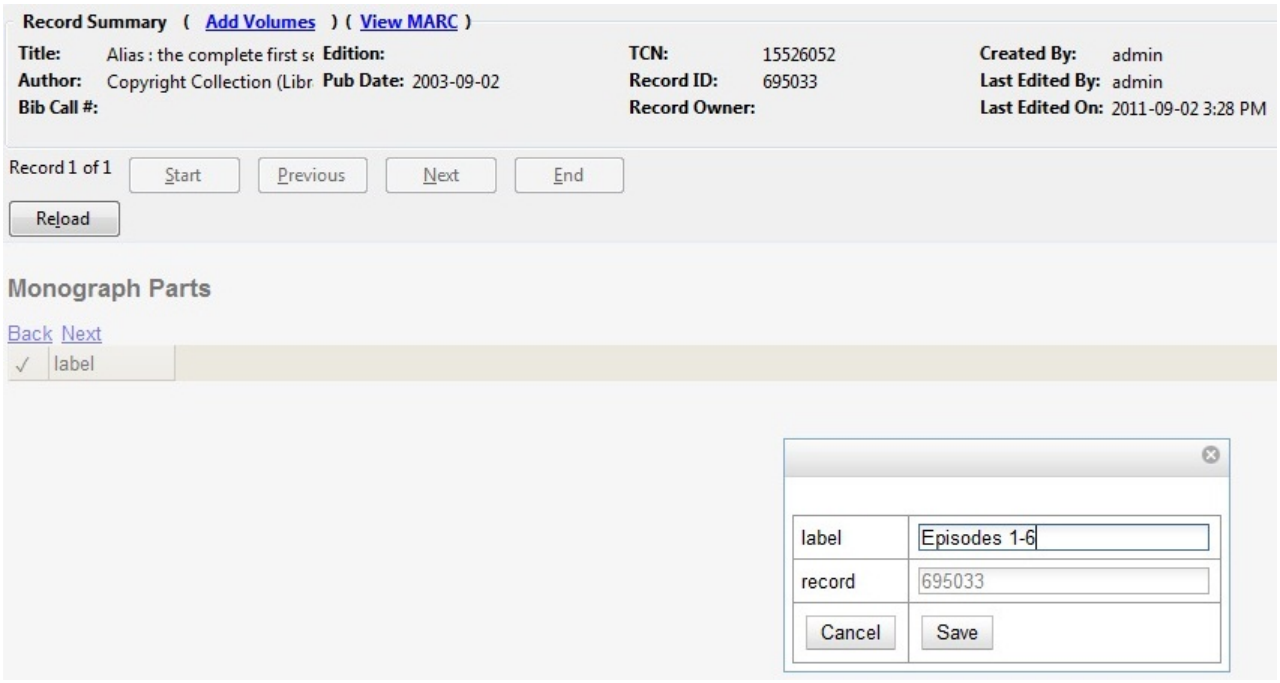
2. Click **Actions for this Record # Manage Parts**



3. Click **New Monograph Part**

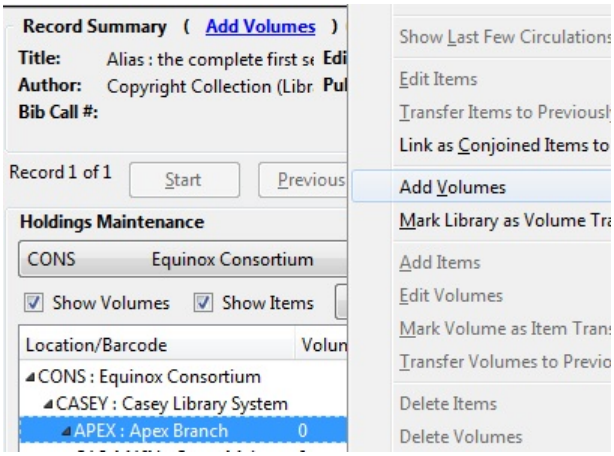
4. Enter the **label** that you want to appear to the user in the catalog, and click **Save**.

   This will create a list of monograph parts from which you can choose when you create a volume and copy.



5. Add a volume and copy. To add a volume and copy to your workstation library, click the **Add Volumes** link in the **Record Summary** at the top of the bibliographic record, or click **Actions for this Record # Add Volumes**.

   To add a volume and copy to your workstation library or other libraries, click **Actions for this Record # Holdings Maintenance # Add Volumes**.

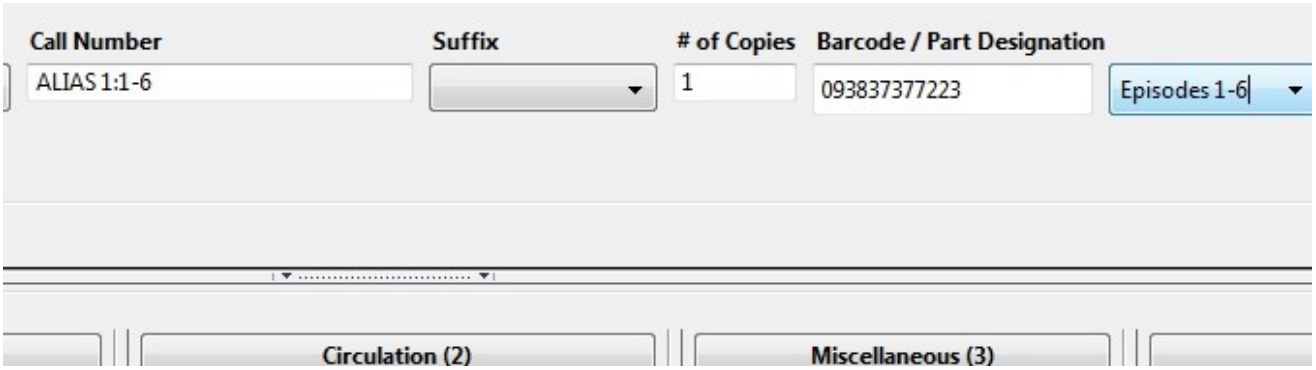


6. The **Unified Volume/Copy Creator** opens. Enter the number of volumes that you want to add to the catalog and the volume description.

7. Enter the number of copies and barcode(s) of each item.

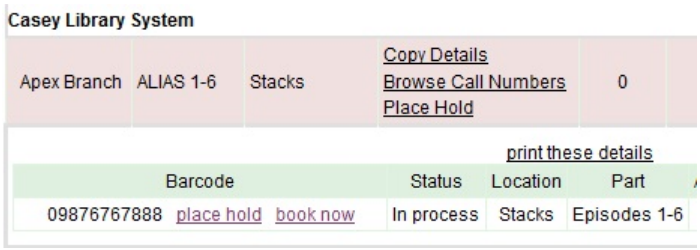
8. Select the **part designation** from the drop down menu adjacent to the barcode field.

9. Apply a template to the copies, or edit fields in the **Copy Editor**.



10.Click **Create Volumes/Items**.

11.The **Holdings Maintenance** screen will refresh to demonstrate the addition of the volume, copy, and part. These fields also appear in the OPAC View.



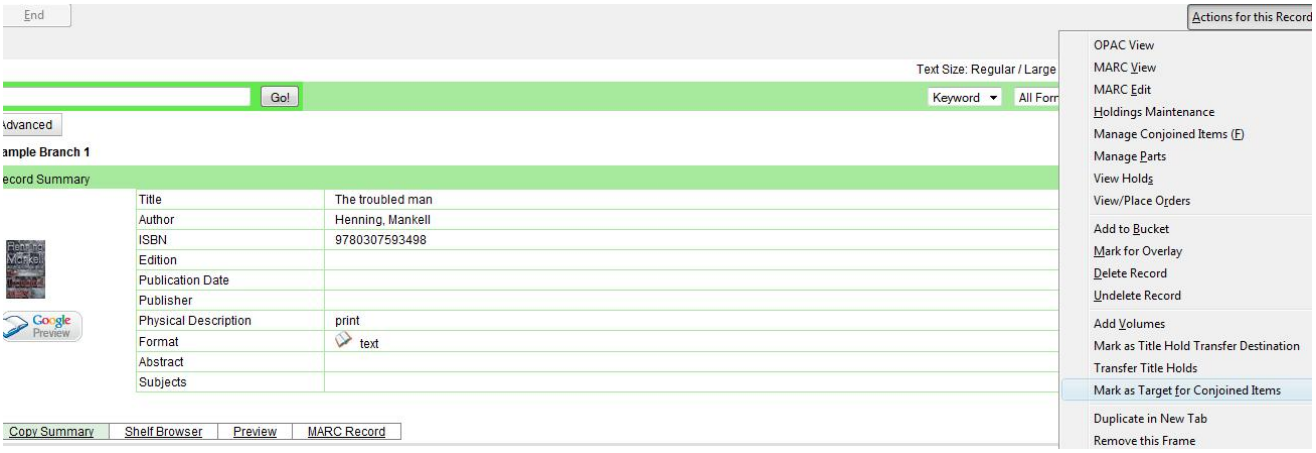
Conjoined Items

Prior to Evergreen version 2.1, items could be attached to only one bibliographic record. The Conjoined Items feature in Evergreen 2.1 enables catalogers to link items to multiple bibliographic records. This feature will enable more precise cataloging. For example, catalogers will be able to indicate items that are printed back to back, are bilingual, are part of a bound volume, are part of a set, or are available as an e-reader pre-load. This feature will also help the user retrieve more relevant search results. For example, a librarian catalogs a multi-volume festschrift. She can create a bibliographic record for the festschrift and a record for each volume. She can link the items on each volume to the festschrift record so that a patron could search for a volume or the festschrift and retrieve information about both works. In the example below, a librarian has created a bibliographic record for two bestselling items. These books are available as physical copies in the library, and they are available as e-reader downloads. The librarian will link the copy of the Kindle to the bibliographic records that are available on the e-reader.

The Conjoined Items feature was designed so that you can link items between bibliographic records when you have the item in hand, or when the item is not physically present. Both processes are described here. The steps are fewer if you have the item in hand, but both processes accomplish the same task. This documentalso demonstrates the process to edit or delete links between items and bibliographic records. Finally, the permission a cataloger needs to use this feature is listed.

Scenario 1: I want to link an item to another bibliographic record, but I do not have the item in hand. 1) Retrieve the bibliographic record to which you would like to link an item.

1. Click **Actions for this Record # Mark as Target for Conjoined Items.**

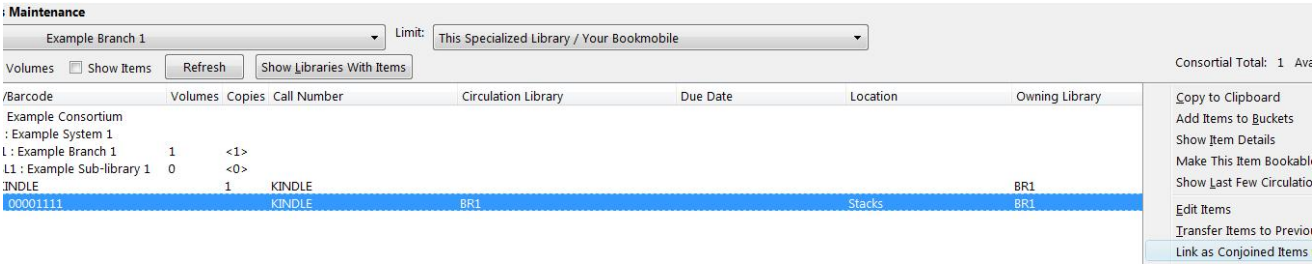


2. A confirmation message will appear. Click **OK.**

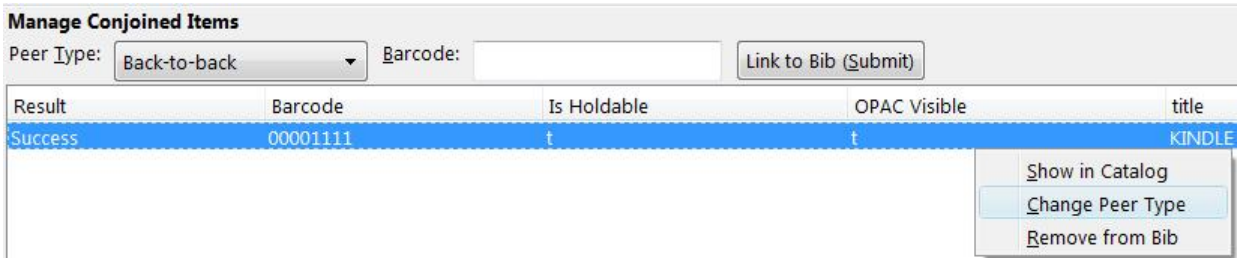
3. In a new tab, retrieve the bibliographic record with the item that you want to link to the other record.

4. Click **Actions for this Record # Holdings Maintenance.**

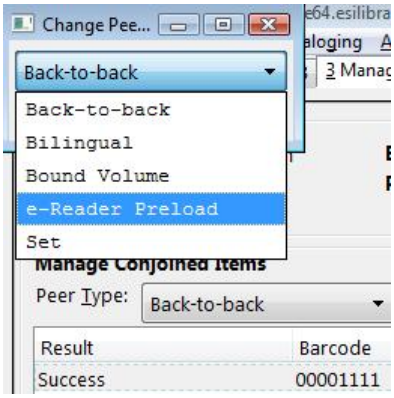
5. Select the copy that you want to link to the other bibliographic record. Right-click, or click **Actions for Selected Rows # Link as Conjoined Items to Previously Marked Bib Record.**



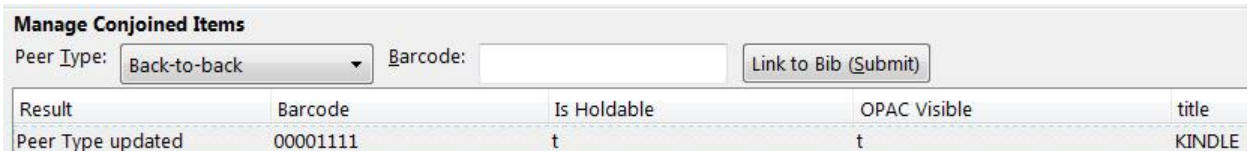
6. The **Manage Conjoined Items** interface opens in a new tab. This interface enables you to confirm the success of the link, and to change the peer type if desired. The **Result** column indicates that you created a successful link between the item and the bib record.



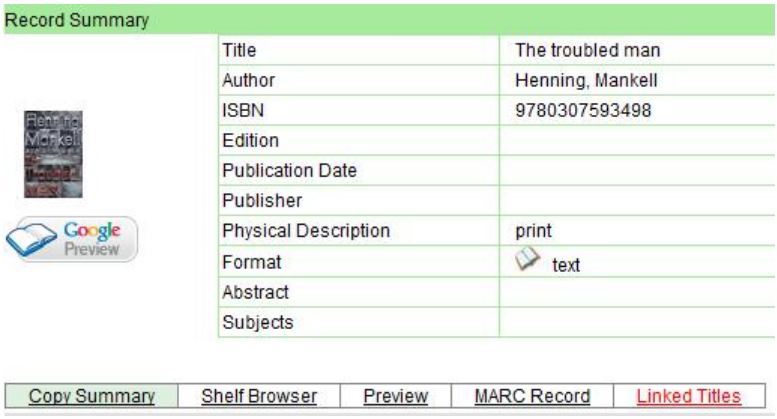
The default peer type, **Back-to-back**, was set as the peer type for our item. To change a peer type after the link has been created, right-click or click **Actions for Selected Items # Change Peer Type**. A drop down menu will appear. Select the desired peer type, and click **OK.**



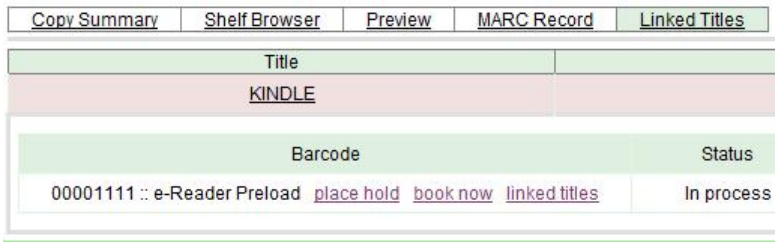
7. The **Result** column will indicate that the **Peer Type** [has been] **Updated.**



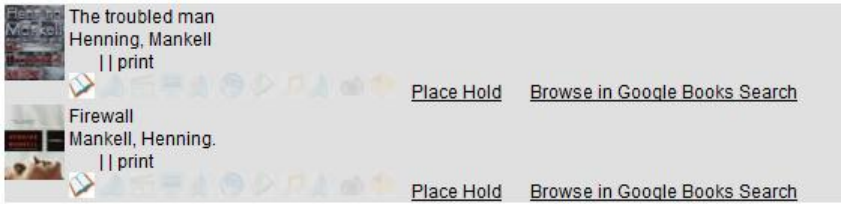
8. To confirm the link between the item and the desired bib record, reload the tab containing the bib record to which you linked the item. Click the link for **Linked Titles.**



9. To view the copy details, including the peer type, click **Copy Details.**

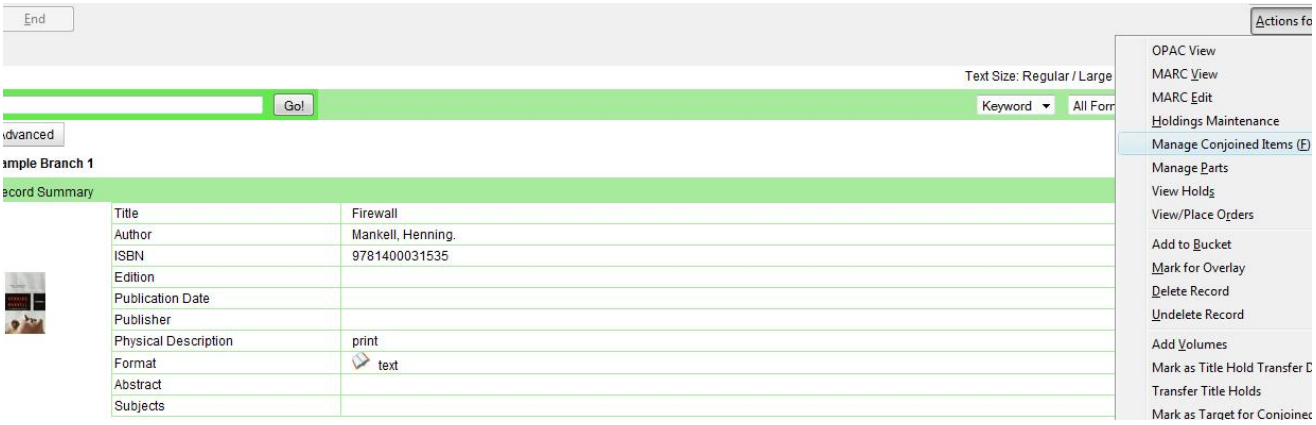


Items can be linked to multiple bibliographic records simultaneously. If you click the linked titles button in the copy details, then you will retrieve a list of bibliographic records to which this item is linked.



Scenario 2: I want to link an item to another bibliographic record, and I do have the item in hand. 1) Retrieve the bibliographic record to which you would like to add the item.

1. Click **Actions for this Record # Manage Conjoined Items.**



2. A note in the bottom left corner of the screen will confirm that the record was targeted for linkage with conjoined items, and the **Manage Conjoined Items** screen will appear.

3. Select the peer type from the drop down menu, and scan in the barcode of the item that you want to link to this record.

4. Click **Link to Bib (Submit).**

5. The linked item will appear in the screen. The **Result** column indicates Success.

6. To confirm the linkage, click **Actions for this Record # OPAC View.**

7. When the bibliographic record appears, click **Reload. Linked Titles** will show the linked title and item.

Scenario 3: I want to edit or break the link between a copy and a bibliographic record. 1) Retrieve the bibliographic record that has a copy linked to it.

8. Click **Actions for this Record # Manage Conjoined Items.**

9. Select the copy that you want to edit, and right-click or click **Actions for Selected Items.**

10. Make any changes, and click **OK.**

Required permission: UPDATE_COPY - Link items to bibliographic records

Call Number Prefixes and Suffixes

In Evergreen version 2.1, you can configure call number prefixes and suffixes in the Admin module. This feature ensures more precise cataloging because each cataloger will have access to an identical drop down menu of call number prefixes and suffixes that are used at his library. In addition, it may streamline cataloging workflow. Catalogers can use a drop down menu to enter call number prefixes and suffixes rather than entering them manually.

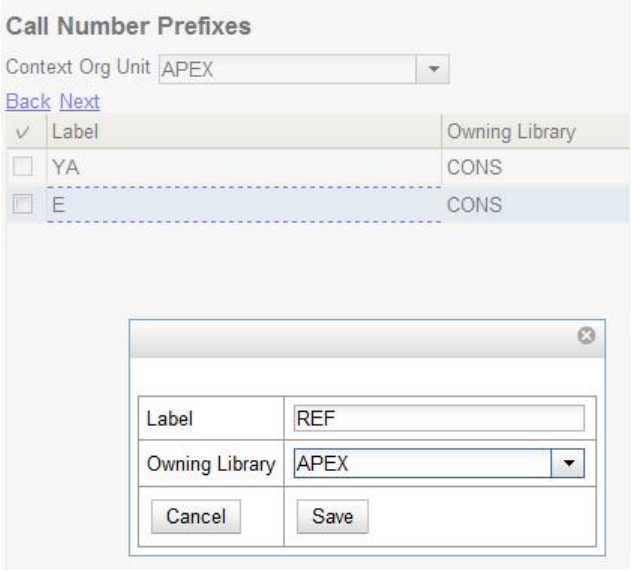
You can also run reports on call number prefixes and suffixes that would facilitate collection development and maintenance.

Configure call number prefixes:

Call number prefixes are codes that precede a call number.

To configure call number prefixes:

1. Select **Admin # Server Administration # Call Number Prefixes**.
2. Click **New Prefix**.
3. Enter the **call number label** that will appear on the item.
4. Select the **owning library** from the drop down menu. Staff at this library, and its descendant org units, with the appropriate permissions, will be able to apply this call number prefix.
5. Click **Save**.

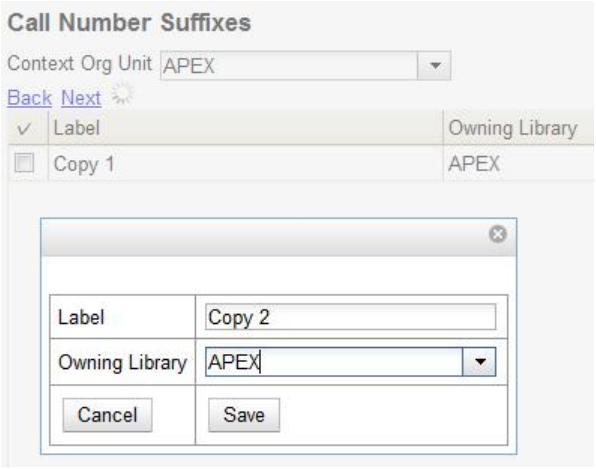


Configure call number suffixes:

Call number suffixes are codes that succeed a call number.

To configure call number suffixes:

1. Select **Admin # Server Administration # Call Number Suffixes**.
2. Click **New Suffix**.
3. Enter the **call number label** that will appear on the item.
4. Select the **owning library** from the drop down menu. Staff at this library, and its descendant org units, with the appropriate permissions, will be able to apply this call number suffix.
5. Click **Save**.



Apply Call Number Prefixes and Suffixes

You can apply call number prefixes and suffixes to items from a pre-configured list in the **Unified Volume/Copy Creator**. See the section, Unified Volume Copy Creator, for an example.

Chapter 7. Using the Booking Module

Abstract

The following chapter will help staff create reservations for cataloged and non- bibliographic items; create pull lists for reserved items; capture resources; and pick up and return reservations.

Creating a Booking Reservation

Only staff members can create reservations. To initiate a reservation, staff can

- search the catalog,
- enter a patron record,
- or use the booking module.

Search the catalog to create a reservation

1. In the staff client, select Search → Search the Catalog
2. Search for the item to be booked.
3. Click Submit Search.
4. A list of results will appear. Select the title of the item to be reserved.
5. After clicking the title, the record summary appears. Beneath the record summary, the copy summary will appear. In the Actions column, select Copy Details.
6. The Copy Details will appear in a new row. In the barcode column, click the book now link.
7. A screen showing the title and barcodes of available copies will appear.
8. Enter the user's barcode in the Reserve to patron barcode box. If the patron barcode does not exist, a pop up box will appear to alert you to the error. After entering the patron's barcode, the user's existing reservations will appear at the bottom of the screen.
9. To the right, a section titled, I need this resource... will allow you to set the dates and times for which the item should be reserved. If the date/time boxes appear in red, then the date and time set is incorrect. For example, if the time for which the reservation is set has already passed, the boxes will appear in red. The times must be set correctly for the reservation to be accomplished. If the item has already been reserved at the time for which you are trying to reserve the item, then you will receive an error message.
10. Finally, select the barcode of the item that you want to reserve. If multiple copies of the item exist, choose the barcode of the copy that you want to reserve, and click Reserve Selected. If you do not select a barcode, and you click Reserve Selected, you will receive an error message. If you do not have a preference, you do not have to select a barcode, and you may click Reserve Any. One of the barcodes will be pulled from the list.

An item must have a status of available or reshelving in order to be targeted for a reservation. If the item is in another status, the reservation will fail.

11. After you have made the reservation, a message will confirm that the action succeeded. Click OK.

12. The screen will refresh, and the reservation will appear below the user's name.

Enter a patron's record to create a reservation

1. Enter the barcode or patron information, and click Search to retrieve the patron's record.

2. The match(es) should appear in the right pane. Click the desired patron's name. In the left panel, a summary of the patron's information will appear. Click the Retrieve Patron button in the right corner to access more options in the patron's record.

3. Eight buttons will appear in the top right corner. Select Other → Booking to create, cancel, pick up, and return reservations.

4. The Copy Details will appear in a new row. In the barcode column, click the book now link.

5. A screen showing the title and barcodes of available copies will appear.

6. Enter the user's barcode in the Reserve to patron barcode box. If the patron barcode does not exist, a pop up box will appear to alert you to the error. After entering the patron's barcode, the user's existing reservations will appear at the bottom of the screen.

7. To the right, a section titled, I need this resource... will allow you to set the dates and times for which the item should be reserved. If the date/time boxes appear in red, then the date and time set is incorrect. For example, if the time for which the reservation is set has already passed, the boxes will appear in red. The times must be set correctly for the reservation to be accomplished. If the item has already been reserved at the time for which you are trying to reserve the item, then you will receive an error message.

8. Finally, select the barcode of the item that you want to reserve. If multiple copies of the item exist, choose the barcode of the copy that you want to reserve, and click Reserve Selected. If you do not select a barcode, and you click Reserve Selected, you will receive an error message. If you do not have a preference, you do not have to select a barcode, and you may click Reserve Any. One of the barcodes will be pulled from the list.

An item must have a status of available or reshelving in order to be targeted for a reservation. If the item is in another status, the reservation will fail.

9. After you have made the reservation, a message will confirm that the action succeeded. Click OK.

10. The screen will refresh, and the reservation will appear below the user's name.

Use the booking module to create a reservation

1. Select Booking → Create or Edit Reservations

2. Enter the barcode of the item and click Next.

3. A screen showing the name of the available resource will appear.

4. Enter the user's barcode in the Reserve to patron barcode box. If the patron barcode does not exist, a pop up box will appear to alert you to the error. After entering the patron's barcode, the user's existing reservations will appear.

5. To the right, a section titled, I need this resource... will allow you to set the dates and times for which the item should be reserved. If the date/time boxes appear in red, then the date and time set is incorrect. For example, if the time for which the reservation is set has already passed, the boxes will appear in red. The times must be set correctly for the reservation to be accomplished. If the resource has already been reserved at the time for which you want to reserve the item, then the item will disappear.

6. Finally, select the resource that you want to reserve. If multiple items or rooms exist, choose the resource that you want to reserve, and click Reserve Selected. If you do not select a resource, and you click Reserve Selected, you will receive an error message. If you do not have a preference, you may click Reserve Any, and one of the resources will be pulled from the list.

7. After you have made the reservation, a message will confirm that the action succeeded. Click OK.

8. The screen will refresh, and the reservation will appear below the user's name.

Cancelling a Reservation

Staff members can cancel a patron's reservation through the Create or Cancel Reservations tab available in a patron's record. Staff members can also cancel a reservation immediately after it has been made.

Enter the patron's record to cancel a reservation

1. Search for and retrieve a patron's record.

2. Select Other → Booking → Create or Cancel Reservations.

3. The existing reservations will appear at the bottom of the screen.

4. To cancel a reservation, highlight the reservation that you want to cancel. Click Cancel Selected.

5. A pop-up window will confirm that you cancelled the reservation. Click OK.

6. The screen will refresh, and the cancelled reservation will disappear.

7. To the right, a section titled, I need this resource... will allow you to set the dates and times for which the item should be reserved. If the date/time boxes appear in red, then the date and time set is incorrect. For example, if the time for which the reservation is set has already passed, the boxes will appear in red. The times must be set correctly for the reservation to be accomplished. If the item has already been reserved at the time for which you are trying to reserve the item, then you will receive an error message.

Cancel a reservation immediately after it has been made

1. Create the reservation.

2. Follow steps four through six in the section, Enter the patron's record to cancel a reservation, to cancel the reservation.

3. The existing reservations will appear at the bottom of the screen.

Creating a Pull List

Staff members can create a pull list to retrieve items from the stacks.

1. To create a pull list, select Booking → Pull List.
2. To find a pull list for your library, select a library from the dropdown box adjacent to See pull list for library.
3. You can decide how many days in advance you would like to select reserved items. Enter the number of days in the box adjacent to Generate list for this many days hence. For example, if you would like to pull items that are needed today, you can enter **1** in the box, and you will retrieve items that need to be pulled today.
4. Click Fetch to retrieve the pull list.
5. The pull list will appear. Click Print to print the pull list.

Capturing Items for Reservations

Staff members can capture items for reservations.

1. In the staff client, select Booking → Capture Resources.
2. Enter the barcode of the items to be captured. Click Capture.
3. A Capture Succeeded message will appear to the right. Information about the item will appear below the message. You can print this information as a receipt and add it to the item if desired.

Picking Up Reservations

Staff members can help users pick up their reservations.

1. In the staff client, select Booking → Pick Up Reservations
2. Enter the user's barcode. Click Go.
3. The title available for pickup will appear. Highlight the title of the item to pick up, and click Pick Up.
4. The screen will refresh to show that the patron has picked up the reservation.

Returning Reservations

Staff members can help users return their reservations.

1. In the staff client, select Booking → Return Reservations.
2. You can return the item by patron or item barcode. Choose Resource or Patron, enter the barcode, and click Go.
3. A pop up box will tell you that the item was returned. Click OK.

4. The screen will refresh to show the reservations that remain out and the resources that have been returned.

Chapter 8. The Serials Module

This documentation is intended for users who will be ordering subscriptions, distributing issues, and receiving issues in Evergreen . 0. Specifically, this tutorial documents the functionality in the serials module and illustrates a basic serials workflow in which the user will register a subscription to a serial publication; distribute issues of that publication to branches; define the captions to be affixed to each issue; specify details of the publication pattern; predict future issues, and receive copies of an issue. Claiming serials is not available in . 0. This document also includes a list of administrative permissions that users must have to use the serials module.

Serial Control View, Alternate Serial Control View, and MFHD Records: A Summary

Serial Control View and Alternate Serial Control View offer you two views of Serials. Both views enable you to create subscriptions, add distributions, define captions, predict future issues, and receive items. Serial Control View was designed for users who work with a smaller number of issues and was designed to accommodate workflows in academic and special libraries. Alternate Serial Control View was designed for users who receive a larger number of issues and was designed for use in public libraries.

The views are interoperable, but because the views were designed for different purposes, some differences emerge. For example, Serial Control View enables you to create and edit serials in a single tabbed interface while Alternate Serial Control View leads you through a series of steps on multiple screens. In addition, receiving functions vary between views. Both receiving interfaces enable you to batch receive issues. However, the Serials Batch Receive interface, which is associated with Alternate Serial Control View, allows for more customization of each receiving unit while the Items tab in Serial Control View allows for greater flexibility in creating multi-issue units, such as in binding serials.

MFHD records that you created in . 6 will also exist in . 0. Pre-existing MFHD records will display above the holdings summary for serials created in Alternate Serial Control View. See simplesect . . 2 for an example of this display. If you create a serial in Serial Control View, the generated holdings and the previous MFHD record will display in a single holdings summary, separated by a comma. You can also create new MFHD records manually.

Copy Templates for Serials

A copy template enables you to specify item attributes that should be applied by default to copies of serials. You can create one copy template and apply it to multiple serials. You can also create multiple copy templates. Templates will be used in the Alternate Serial Control View or the Serial Control View.

Create a copy template

1. To create a copy template, click Admin # Local Administration # Copy Template Editor.

2. Enter a Name for the template.

3. Select an owning library from the Owning lib drop down menu. This organization owns the copy template. A staff member with permissions at that organization can modify the copy template. The menu is populated from the organizations that you created in Admin # Server Administration # Organizational Units.

4. Click the box adjacent to Circulate? If you want the item to circulate.

5. Check the box adjacent to Holdable? if patrons can place holds on the item.

6. Check the box adjacent to OPAC Visible? if you want patrons to be able to see the item in the OPAC after you receive it.

7. Select a loan duration rule from the drop down menu.

8. Select a fine level for the item from the drop down menu.

9. Select a copy Location from the drop down menu. The menu is populated from the copy locations that you created in Admin # Local Administration # Copy Locations.

10. Select a circ modifier from the drop down box. The menu is populated from the modifiers that you created in Admin # Server Administration # Circulation Modifiers.

11. Check the box adjacent to Floating? if the item is part of a floating collection.

12. Check the box adjacent to Deposit? if patrons must place a deposit on the copy before they can use it.

13. Check the box adjacent to Reference? if the item is a reference item.

14. If the item is in mint condition, then check the box adjacent to Mint Condition?

15. Enter age protection rules in the Age Protect field. Age protection allows you to control the extent to which an item can circulate after it has been received. For example, you may want to protect new copies of a serial so that only patrons who check out the item at your branch can use it.

16. Enter a message in the Alert Message field. This message will appear every time the item is checked out to a patron.

17. Enter a code from the MARC fixed fields if you want to control the circulation based on the item type in the Circ as Type field.

18. Enter a deposit amount if patrons must place a deposit on the copy before they can use it.

19. Enter the price of the item.

20. Enter the ID of the copy status in the Status field. A list of copy statuses and their IDs can be found in Admin # Server Administration # Copy Status.

21. Click Save.

Fine level and loan duration are required fields in the Copy Template Editor.

Edit a copy template

You can make changes to an existing copy template. Changes that you make to a copy template will apply to any items that you receive after you edited the template.

1. To edit a copy template, click your cursor in the row that you want to edit. The row will turn blue.

2. Double-click. The copy template will appear, and you can edit the fields.

3. After making changes, click Save.

From the copy template interface, you can delete copy templates that have never been used.

Alternate Serial Control View

Using the Alternate Serial Control View, you can create a subscription, a distribution, a stream, and a caption and pattern, and you can generate predictions and receive issues. To access Alternate Serial Control View, open a serials record, and click Actions for this Record # Alternate Serial Control. This opens the Subscriptions interface

Subscriptions

Add new subscriptions to a serials record that exists in the catalog.

Create a subscription

1. Click New Subscription.

2. Select an owning library. The owning library indicates the organizational unit(s) whose staff can use this subscription. This menu is populated with the shortnames that you created for your libraries in the organizational units tree in Admin # Server Administration # Organizational Units. The rule of parental inheritance applies to this list. For example, if a system is made the owner of a subscription, then users, with appropriate permissions, at the branches within the system could also use this subscription.

3. Enter the date that the subscription begins in the start date. Recommended practice is that you select the date from the drop down calendar although you can manually enter a date. Owning library and start date are required fields in the new subscription pop up box.

4. Enter the date that the subscription ends in the end date. Recommended practice is to select a date from the drop down calendar, but you can manually enter a date, also.

5. Enter the difference between the nominal publishing date of an issue and the date that you expect to receive your copy in the Expected Date Offset. For example, if an issue is published the first day of each month, but you receive the copy two days prior to the publication date, then enter -2 days into this field.

6. Click Save.

7. After you save the subscription, it will appear in a list with a hyperlinked ID number. Use the drop down menu at the top of the screen to view subscriptions at other organizations.

Manage a subscription

Click the hyperlinked ID number to manage the subscription. The tabbed interface enables you to create distributions, captions and patterns, and issuances.

Edit a subscription

Edit a subscription as you would edit a copy template.

Distributions

Distributions indicate the branches that should receive copies of a serial. Distributions work together with streams to indicate the number of copies that should be sent to each branch.

Create a distribution

1. Click the Distributions tab.
2. Click New Distribution.
3. Enter a name for the distribution in the Label field. It may be useful to identify the branch to which you are distributing these issues in this field. This field is not publicly visible and only appears when an item is received. There are no limits on the number of characters that can be entered in this field.
4. Select a holding library from the drop down menu. The holding library is the branch that will receive the copies.
5. Select a copy template from the Receive Unit Template drop down menu. This menu is populated with the copy templates that you created in Copy Template Editor.

Label, Holding Library, and Receive Unit Template are required fields in the new distribution pop up box.

6. Ignore the fields, Unit Label Prefix and Unit Label Suffix. These fields are not functional in Alternate Serial Control View.
7. Click Save. The distribution will appear in a list in the Distributions tab in the Subscription Details.

Edit a distribution

Edit a distribution just as you would edit a copy template.

From the distribution interface, you can also delete distributions. Deleting the distribution would delete related data, such as streams associated with this distribution, but it would not delete units, the copy-equivalent objects that hold barcodes. Recommended practice is that you do not delete distributions.

Streams

Distributions work together with streams to indicate the number of copies that should be sent to each branch. Distributions identify the branches that should receive copies of a serial. Streams identify how many copies should be sent to each branch. Streams are intended for copies that are received on a recurring, even if irregular, basis.

In our example, the Apex Branch should receive copies, so we created a distribution to that branch. The Apex Branch should receive two copies, so we will create two streams to that branch.

Create a stream

Click the hyperlinked title of the distribution. The number of streams that have already been created for this distribution displays adjacent to the title. You can choose one of two ways to create a stream: New Stream or Create Many Streams. The New Stream button allows you to create one new stream and assign it a routing label.

1. Click New Stream

2. Enter a routing label so that the copy could be read by specific users or departments before the copy is shelved. The routing label appears during receiving and could be added to routing lists; it is not viewable by the public. Routing lists do not print from in 2.0. This field is optional.

3. Click Save.

The Create Many Streams button allows you to create multiple streams at once, but it does not allow you to add a routing label when you create the stream.

1. Click Create Many Streams.

2. Enter the number of streams that you want to create in the How many? Field.

3. Click Create.

Edit a stream

Edit a stream just as you would edit a copy template.

From the streams interface, you can also delete streams. Deleting the stream would delete related data, but it would not delete units, or the copy-equivalent objects that hold barcodes. Recommended practice is that you do not delete streams.

Captions and Patterns

The Captions and Patterns wizard allows you to enter caption and pattern data as it is described by the 853, 854, and 855 MARC tags. These tags allow you to define how issues will be captioned, and how often the library receives issues of the serial.

In 2.0, it is not possible to create a caption and pattern and apply it to multiple subscriptions. However, you can re-use patterns if you copy and paste to and from the pattern code field in the Captions and Patterns tab.

Create a Caption and Pattern

1. Open the Subscription Details.

2. Click the Captions and Patterns tab.

3. Click Add Caption and Pattern.

4. In the Type drop down box, select the MARC tag to which you would like to add data.

5. In the Pattern Code drop down box, you can enter a JSON representation of the 85X tag by hand, or you can click the Wizard to enter the information in a user-friendly format.

6. The Caption and Pattern that you create is Active by default, but you can deactivate a caption and pattern at a later time by unchecking the box.

A subscription may have multiple captions and patterns listed in the subscripiton details, but only one Caption and Pattern can be active at any time. If you want to add multiple patterns, e.g. for Basic and Supplement, Click Add Caption and Pattern.

Use the Pattern Code Wizard

The Pattern Code Wizard enables you to create the caption of the item and add its publication information. The Wizard is composed of five pages of questions. You can use the Next and Previous navigation buttons in the top corners to flip between pages.

1. To add a pattern code, click Wizard.

2. Page 1: Enumerations

    1. To add an enumeration, check the box adjacent to Use enumerations?. The enumerations conform to $a-$h of the 853,854, and 855 MARC tags.

    2. A field for the First level will appear. Enter the enumeration for the first level. A common first level enumeration is volume, or “v.”

    3. Click Add Enumeration.

    4. A field for the Second level will appear. Enter the enumeration for the second level. A common first level enumeration is number, or “no.”

    5. Enter the number of bibliographic units per next higher level. This conforms to $u in the 853, 854, and 855 MARC tags.

    6. Choose the enumeration scheme from the drop down menu. This conforms to $v in the 853, 854, and 855 MARC tags.

        You can add up to six levels of enumeration.

    7. Add Alternate Enumeration if desired.

    8. When you have completed the enumerations, click Next.

3. Page 2: Calendar

    1. To use months, seasons, or dates in your caption, check the box adjacent to Use calendar changes?

    2. Identify the point in the year at which the highest level enumeration caption changes.

    3. In the Type drop down menu, select the points during the year at which you want the calendar to restart.

4. In the Point drop down menu, select the specific time at which you would like to change the calendar
5. To add another calendar change, click Add Calendar Change. There are no limits on the number of calendar changes that you can add.
6. When you have finished the calendar changes, click Next.

4. Page 3: Chronology
    1. To add chronological units to the captions, check the box adjacent to Use chronology captions?
    2. Choose a chronology for the first level. If you want to display the terms, “year” and “month” next to the chronology caption in the catlaog, then check the box beneath Display in holding field?
    3. To include additional levels of chronology, click Add Chronology Caption. Each level that you add must be smaller than the previous level.
    4. After you have completed the chronology caption, click Next.
5. Page 4: Compress and Expand Captions
    1. Select the appropriate option for compressing or expanding your captions in the catalog from the compressibility and expandability drop down menu. The entries in the drop down menu correspond to the indicator codes and the subfield $w in the 853 tag. Compressibility and expandability correspond to the first indicator in the 853 tag.
    2. Choose the appropriate caption evaluation from the drop down menu.
    3. Choose the frequency of your publication from the drop down menu. For irregular frequencies, you may wish to select use number of issues per year, and enter the total number of issues that you receive each year. However, in the . 0 release, recommended practice is that you use only regular frequencies. Planned development will create an additional step to aid in the creation of irregular frequencies.
    4. Click Next.
6. Page 5: Finish Captions and Patterns
    1. To complete the wizard, click Create Pattern Code.
    2. Return to Subscription Details.
    3. Confirm that the box adjacent to Active is checked. Click Save Changes. The row is now highlighted gray instead of orange.

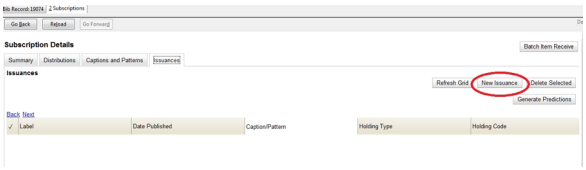
Issuances

The Issuances tab enables you to manually create an issue in the ILS. The ILS will use the initial issue that you manually create to predict future issues.

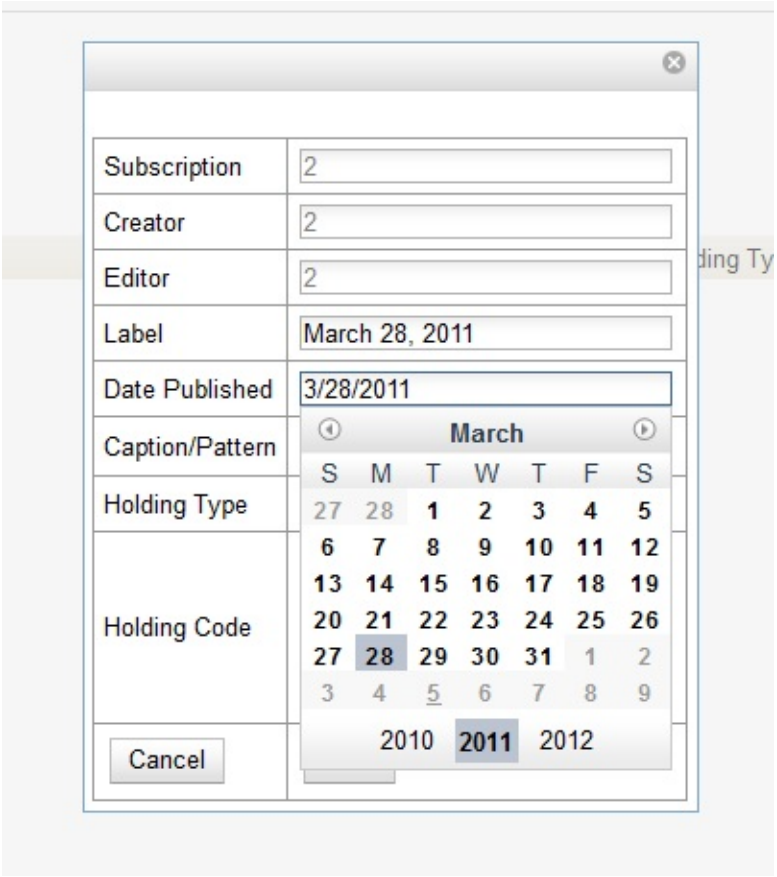
Create an issuance

1. Click the Issuances tab in the Subscription Details.

2. Click New Issuance.



3. The Subscription, Creator, and Editor fields contain subscription and user IDs, respectively. These fields are disabled because Evergreen automatically fills in these fields.



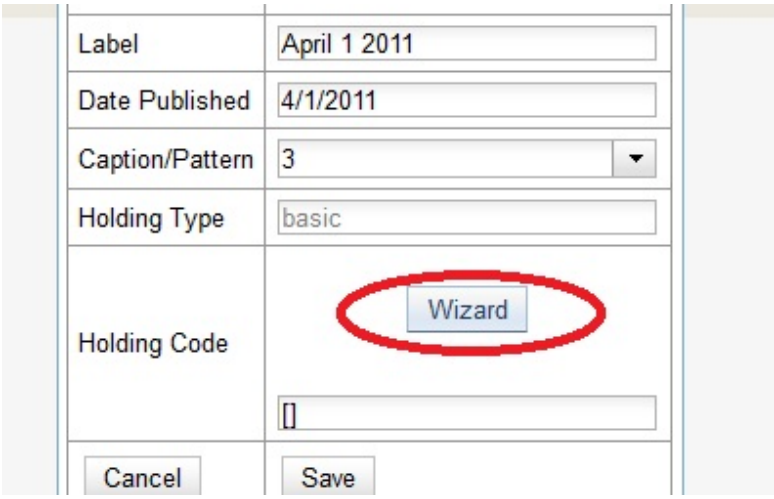
4. Enter a name for this issuance in the Label field. There are no limits on the number of characters that can be entered in this field. You may want to enter the month and year of the publication in hand.

5. Enter the Date Published of the issuance that you are editing. Recommended practice is that you select the date from the drop down calendar although you can manually enter a date. If you are creating one manual issue before automatically predicting more issues, then this date should be the date of the most current issue before the prediction starts.

6. Select a Caption/Pattern from the drop down menu. The numbers in the drop down menu correspond to the IDs of the caption/patterns that you created.

7. The Holding Type appears by default and corresponds to the Type that you selected when you created the Caption/Pattern.

8. In the holding code area of the New Issuance dialog, click Wizard. The Wizard enables you to add holdings information.



9. Enter the volume of the item in hand in the v. field.

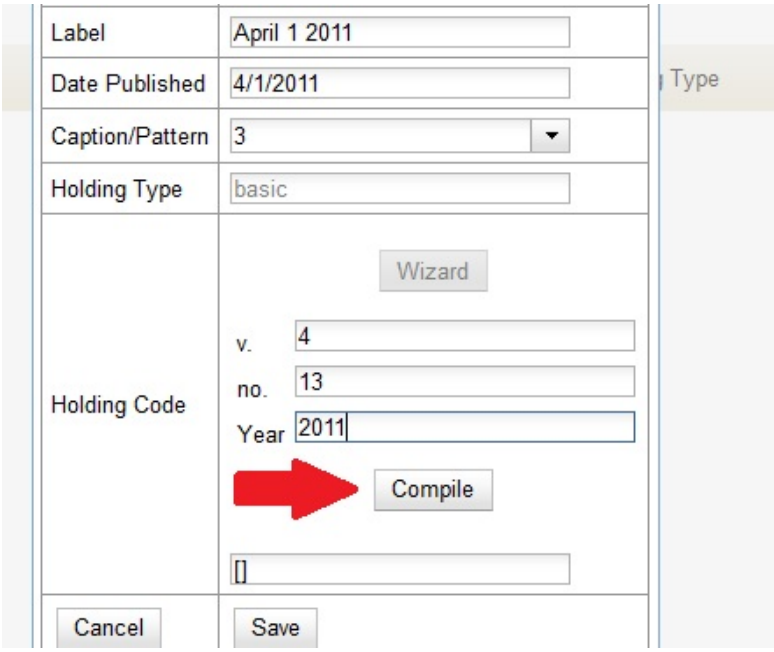
10. Enter the number of the item in hand in the no. field.

11. Enter the year of publication in the Year field.

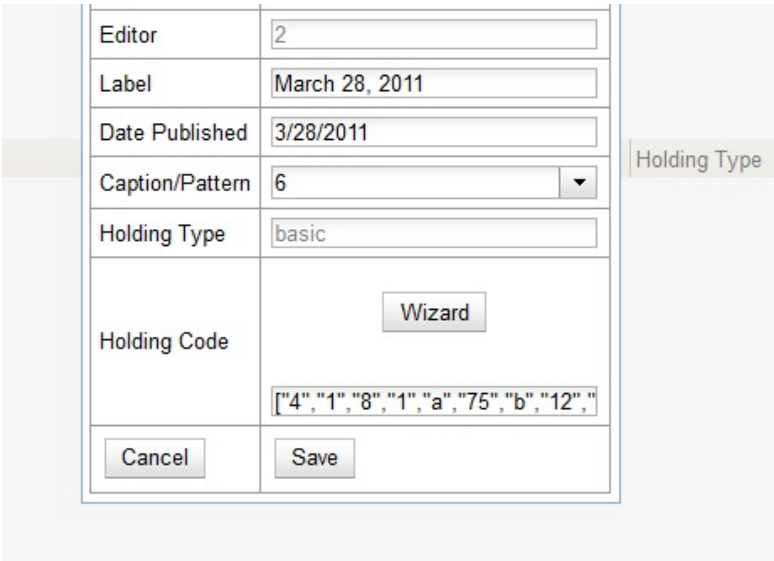
12. Enter the month of publication in the Month field if applicable. You must enter the calendar number of the month rather than the name of the month. For example, enter 12 if the item in hand was published in December.

13. Enter the day of publication in the day field if applicable.

14. Click Compile to generate the holdings code.



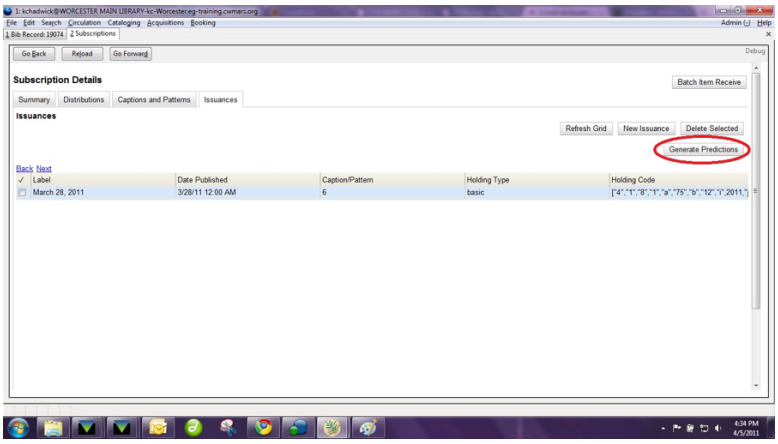
15. Click Save. The newly generated issuance will appear in a list in the Issuances tab of the Subscription Details.



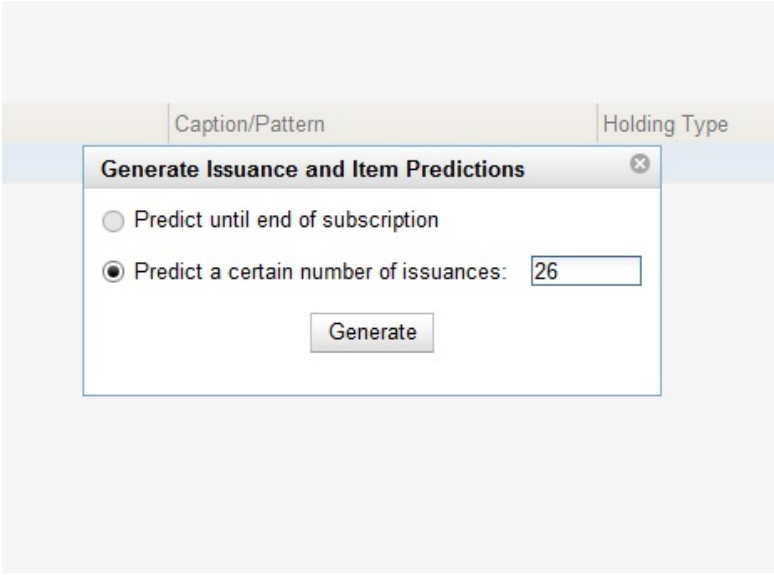
Generate item predicitions

After you manually create the first issue, Evergreen will predict future issuances. Use the Generate Predictions functionality to predict future issues.

1. Click Subscription Details # Issuances # Generate Predictions.

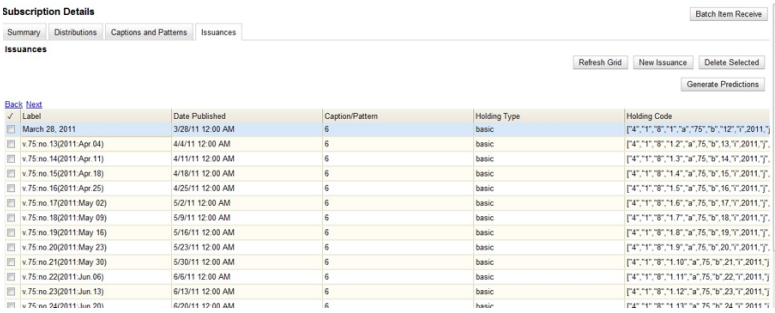


2. Choose the length of time for which you want to predict issues. If you select the radio button to predict until end of subscription, then Evergreen will predict issues until the end date that you created when you created the subscription. See simplesect . 1 for more information. If you do not have an end date, select the radio button to predict a certain number of issuances, and enter a number in the field.



3. Click Generate.

4. Evergreen will predict a run of issuances and copies. The prediction will appear in a list.



5. You can delete the first, manual issuance by clicking the check box adjacent to the issuance and clicking Delete Selected.

Receiving

You can batch receive items through a simple or an advanced interface. The simple interface does not allow you to add barcodes or use the copy template. These items are also not visible in the OPAC. The advanced interface enables you to use the copy templates that you created, add barcodes, and make items OPAC visible and holdable.

You can access both Batch Receive interfaces from two locations in the ILS. From the Subscription Details screen, you can click Batch Item Receive. You can also access these interfaces by opening the catalog record for the serial, and clicking Actions for this Record # Serials Batch Receive.

Simple Batch Receiving

Follow these steps to receive items in batch in a simple interface.

1. The Batch Receive interface displays issues that have not yet been received. The earliest expected issue appears at the top of the list.

2. In the right lower corner, you see a check box to Create Units for Received Items. If you do not check this box, then you will receive items in simple mode.

3. Click Next.

4. In simple mode, the distributions that you created are displayed. They are marked received by default. If you hover over the branch name, you can view the name of the distribution and its stream.

5. You can receive and add a note to each item individually, or you can perform these actions on all of the distributions and streams at once. To do so, look above the line, and enter the note that you want to apply to all copies and confirm that the box to Receive? is checked.

6. Click Apply. The note should appear in the note field in each distribution.

In 2.0, the note field is only displayed in the current screen.

7. Then click Receive Selected Items.

8. The received items are cleared from the screen.

Advanced Batch Receiving

Follow these steps to receive items in batch in a simple interface.

1. The Batch Receive interface displays issues that have not yet been received. The earliest expected issue appears at the top of the list.

2. If you want to barcode each copy, display it in the catalog, and make it holdable, then check the box adjacent to Create Units for Received Items in the lower right side of the screen.

3. This will allow you to utilize the copy templates and input additional information about the copy:

    1. Barcode – You can scan printed barcodes into the barcode field for each copy, or you can allow the system to auto-generate barcodes. …To auto-generate barcodes, check the box adjacent to Auto-generate?, and enter the first barcode into the barcode field in the first row of the table. Then press the Tab key. The remaining barcode fields will automatically populate with the next barcodes in sequence, including check digits.

    2. Circ Modifiers - The circ modifiers drop down menu is populated with the circulation modifiers that you created in Admin # Server Administration # Circulation Modifiers. If you entered a circ modifier in the copy template that you created for this subscription, then it will appear by default in the distributions.

    3. Call Number – Enter a call number. Any item with a barcode must also have a call number.

    4. Note – Add a note. There are no limits on the number of characters that can be entered in this field. The note only displays in this screen.

    5. Copy Location – The copy location drop down menu is populated with the copy locations that you created in Admin # Local Administration # Copy Location Editor. If you entered a copy location in the copy template that you created for this subscription, then it will appear by default in the distributions.

6. Price - If you entered a price in the copy template that you created for this subscription, then it will appear by default in the distributions. You can also manually enter a price if you did not include one in the copy template.

7. Receive? – The boxes in the Receive? Column are checked by default. Uncheck the box if you do not want to receive the item. Evergreen will retain the unreceived copies and will allow you to receive them at a later time.

4. When you are ready to receive the items, click Receive Selected Items.

5. The items that have been received are cleared from the Batch Receive interface. The remaining disabled item is an unreceived item.

6. If the items that you received have a barcode, a copy template that was set to OPAC Visible, and are assigned a shelving location that is OPAC Visible, then you can view the received items in the catalog. Notice that the Holdings Summary has been updated to reflect the most recent addition to the holdings.

Serial Control View

Serial Control View is separate from the Alternate Serial Control interface. Serial Control View enables you to manage serials in a single tabbed interface. This view also enables you to bind units. Serial Control View consists of five tabs: Items, Units, Distributions, Subscriptions, and Claims. Units and Claims are not functional in 2.0.

To access Serial Control View, open a bib record and click Actions for this Record # Serial Control View.

Subscriptions

The Subscriptions tab enables you to view and manage subscriptions.

Create a subscription

1. Click the Subscriptions tab.

2. Select the branch that will own the subscription.

3. Right-click or click Actions for Selected Row, and click Add Subscription.

4. Enter the date that the subscription begins in the start date, and click Apply. You must enter the date in YYYY-MM-DD format.

5. Enter the date that the subscription ends in the end date. This field is optional.

6. Enter the difference between the nominal publishing date of an issue and the date that you expect to receive your copy in the Expected Date Offset. For example, if an issue is published the first day of each month, but you receive the copy two days prior to the publication date, then enter -2 days into this field.

7. When finished, click Create Subscription(s) in the bottom right corner of the screen.

8. A confirmation message appears. Click OK.

You can add notes to the subscription by clicking Subscription Notes. These notes are currently viewable only in the staff client by clicking on the Subscription Notes button.

Edit a subscription

To edit a subscription, select the subscription in the tree on the left side of the screen. You can edit the following categories: Owning Lib, Start Date, End Date, and Date Offset. After you edit the subscription, click Modify Subscription(s) to save the changes.

Distributions

Distributions indicate the branches that should receive copies of a serial. Distributions work together with streams to indicate the number of copies that should be sent to each branch.

Create a distribution

1. Click the distributions link beneath the subscription. Right click or click Actions for Selected Rows, and click Add distribution.
2. Apply a new label to the distribution. It may be useful to identify the branch to which you are distributing these issues in this field. This field is not publicly visible and only appears when an item is received. There are no limits on the number of characters that can be entered in this field.
3. Apply a prefix to the spine label if desired. This information will display in Serial Control View when the items are received, but it does not print on the spine label in . 0.
4. Apply a suffix to the spine label if desired. This information will display in Serial Control View when the items are received, but it does not print on the spine label in . 0.
5. The holding library is filled in by default and is the library to which you attached the subscription.
6. The Legacy Record Entry contains the MFHD records that are attached to the bib record if the owning library is identical to the distribution's holding library. A distribution can thus be an extension of an MFHD record. Select the MFHD record from the drop down menu.
7. The Receive Call Number field is empty until you receive the first item. When you receive the first item, you are prompted to enter a call number. That call number will populate this drop down menu.
8. The Bind Call Number field is empty until you bind the first item. When you receive the first item, you are prompted to enter a call number. That call number will populate this drop down menu.
9. Receive Unit Template – The template that should be applied to copies when they are received. Select a template from the drop down menu.
10. Bind Unit Template - The template that should be applied to copies when they are bound. Select a template from the drop down menu.
11. When finished, click Create Distribution(s) in the bottom right corner of the screen.
12. A confirmation message appears. Click OK.

You can add notes to the distribution by clicking Distribution Notes. These notes are currently viewable only in the staff client by clicking on the Distribution Notes button.

Edit a distribution

To edit a distribution, select the distribution in the tree on the left side of the screen. You can edit the following categories: Label, Holding Lib, Legacy Record Entry, Receive Unit Template, Bind Unit Template, Receive Call Number and Bind Call Number. After you edit the distribution, click Modify Distribution(s) to save the changes.

Streams

Distributions work together with streams to indicate the number of copies that should be sent to each branch. Distributions identify the branches that should receive copies of a serial. Streams identify how many copies should be sent to each branch. Streams are intended for copies that are received on a recurring, even if irregular, basis.

In our example, the Apex Branch should receive copies, so we created a distribution to that branch. The Apex Branch should receive two copies, so we will create two streams to that branch.

Create a stream

1. Click the Distributions tab.
2. Check the boxes to Show Dist. and Show Groups to view distributions and streams.
3. Select the Streams link beneath the distribution that you created for that branch. Right click or click Actions for Selected Row # Add Stream.
4. Click the stream that is created.
5. Enter a routing label so that the copy could be read by specific users or departments before the copy is shelved. The routing label appears during receiving and could be added to routing lists; it is not viewable by the public. Routing lists do not print in . 0. This field is optional.
6. Click Modify Stream(s) in the bottom right corner of the screen.

The data in the Basic Summary, Supplement Summary, and Index Summary are automatically generated by the ILS when you create a caption and pattern and a holdings statement. You can create additional textual holdings manually by editing the Textual Holdings field.

Edit a stream

1. To edit a stream, select the stream in the tree on the left side of the screen. You can edit the following category:
   - Routing Label – The label given to an issue to direct it to the people or departments that should view the issue before it is available to the public.
2. The Basic Summary displays the distribution ID, the Textual Holdings, and the Generated Holdings. The OPAC uses data in legacy records, the generated coverage field, and the textual holdings fields to display holdings information.
   1. The distribution ID and the Generated Coverage are created by Evergreen.
   2. Textual Holdings – Enter any additional holdings information in this field, and it will display in the OPAC as Additional Volume Information.

3. Then click Modify Basic Summary to save your changes. Your changes will appear in the OPAC view.

Captions and Patterns

The Captions and Patterns wizard allows you to enter caption and pattern data as it is described by the 853, 854, and 855 MARC tags. These tags allow you to define how issues will be captioned, and how often the library receives issues of the serial.

In 2.0, it is not possible to create a caption and pattern and apply it to multiple subscriptions. However, you can re-use patterns if you copy and paste to and from the pattern code field in the Captions and Patterns tab.

Create a caption and pattern

1. Click the Subscriptions tab.
2. Beneath the subscription, click Captions and Patterns, and right-click or click Actions for Selected Row # Add Caption/Pattern.
3. The ID and Creation Date will fill in automatically.
4. Click the Unset entry beneath Type. A drop down menu will appear. Choose the type of caption and pattern that you want to create, and click Apply.
5. Click the Unset entry beneath Active. A drop down menu will appear. Choose Yes if you want to activate the caption and pattern. Click Apply.
6. Click the Unset entry beneath the Pattern Code (temporary) field if you want to create the pattern code by hand. If you want to create it automatically, click Pattern Code Wizard in the lower right corner.
7. Follow the steps for using the pattern code wizard.
8. Click Apply.
9. Click Create Caption and Pattern(s).

Edit a caption and pattern

To edit a caption/pattern, select the caption/pattern in the tree on the left side of the screen. You can edit the following categories: - Type – Change the type of the caption/pattern. - Acitve – Activate or deactivate the caption/pattern. - Pattern Code – Edit the contents of the field, or click the Pattern Code Wizard to create a new pattern code.

After you edit the subscription, click Modify Caption and Pattern(s) to save the changes.

Issuances

The Issuances tab enables you to manually create an issue in the ILS. The ILS will use the initial issue that you manually create to predict future issues.

Create an issuance

1. Click the Subscriptions tab.

2. Beneath the subscription, click Issuances, and right-click or click Actions for Selected Row # Add Issuance.
3. The fields in the first column will fill in automatically after you have created the issuance.
4. Click the Unset link in the Holding Code field, and manually enter a holding code. Click Apply.
5. Click the Unset link in the Caption/Pattern field. Select a caption/pattern from the drop down menu. Click Apply.
6. Enter the Date Published of the issuance that you are editing. Enter the date in YYYY-MM-DD format. If you are creating one manual issue before automatically predicting more issues, then this date should be the date that you want to enter before the prediction starts. Click Apply.
7. Click in the Issuance Label field to name the issuance. There are no limits on the number of characters that can be entered in this field. You may want to enter the month and year of the publication in hand. Click Apply.
8. Click Create Issuance in the lower right corner to save your changes.
9. A confirmation message appears. Click OK.

Edit an issuance

To edit an issuance, select the issuance in the tree on the left side of the screen. You can edit the following categories: Holding Code, Caption/Pattern, Date Published, and Issuance Label. After you edit the issuance, click Modify Issuance(s) to save the changes.

Generate item predictions

1. Open the Subscriptions tab.
2. Right-click or click Actions for Selected Row # Make predictions.
3. A pop up box will aks you how many items you want to predict. Enter the number, and click OK.
4. A confirmation message will appear. Click OK.
5. Click the Issuances link to view the predicted issues.

Receiving

Receive items in the Items tab. From this interface, you can receive items, edit item attributes, and delete items.

Receive Items

1. To receive items, click the Receive radio button. In the top half of the screen, the items that have yet to be received are displayed. In the bottom half of the screen, recently received items are displayed.
2. Select the branch that will receive the items from the drop down box.
3. Select the issue that you want to receive.
4. Select the current working unit. Click Set Current Unit, located in the lower right corner of the screen. A drop down menu will appear.

- If you want to barcode each item individually, select Auto per item. This setting is recommended for most receiving processes.
- If you want each item within a unit to share the same barcode, then select New Unit. This setting is advised for most binding processes.
- If you want the item to be received or bound into an existing item, select Recent and select the desired issue. To making a change in bound items, receive or bind the items into an already existing unit.

5. Click Receive/Move Selected.
6. Enter a barcode and call number if prompted to do so.
7. A message confirming receipt of the item appears. Click OK.
8. The screen refreshes. In the top half of the screen, the item displays a received date. In the bottom half of the screen, the item that you have just received is now at the top of the list of the received items.

After receiving items, you can view the updated holdings in the OPAC. In this example, the legacy MFHD record and the items recently received in the serial control view display together in the MFHD statement.

Edit Item Attributes

In this pop up box, you can view the Item ID, Status, Distribution, and Shelving ID. These are generated by Evergreen. However, you may need to edit an item's Date Expected or Received.

1. To edit item attributes, select the item(s) that you want to edit, and click Actions for Selected Rows # Edit Item Attributes.
2. Edit the attributes that appear. When you are finished, click Modify Item(s).

Delete Items

You can use this menu item to delete items from your holdings. To delete items from your holdings, click Actions for Selected Rows # Delete Item.

Bind Items

The binding mode applies the binding template, which is defined in the distribution (see simplesect 2 for more information), to units that should be bound.

1. Select the the branch that will receive the items from the drop down box.
2. To bind items, click the Bind radio button. Items that have been received will appear in the top half of the screen.
3. Select the current working unit.
4. Select the issues that you want to bind, and click Receive/Move Selected.
5. In the bottom half of the screen, you can view the items that you have bound together.

If you want to view all items, including those that have not been received, in the top half of the screen, click the check box adjacent to Show All.

MFHD Record

You can manually create MFHD statements.

1. Create an MFHD record
2. Open a serial record, and in the bottom right corner above the copy information, click Add MFHD Record. You can also add the MFHD statement by clicking Actions for this Record #MFHD Holdings #Add MFHD Record.
3. A message will confirm that you have created the MFHD Record. Click OK.
4. Click Reload in the top left corner of the record.
5. The Holdings Summary will appear. Click Edit Holdings in the right corner.
6. Click Edit Record.
7. The MFHD window will pop up. Enter holdings information. Click Save MFHD.
8. Close the MFHD window.
9. Click Reload in the top left corner of the record. The Holdings Summary will reflect the changes to the MFHD statement.

The following permissions enable you to control serials' functions. Although you can assign each permission to users in the Admin module, it is recommended that either all serials permissions be assigned to an individual, or that they should be assigned to individuals in the following groups.

The following permission allow you to create, manage, view, edit, and perform all other functions associated with these serials tasks:

- ADMIN_SERIAL_CAPTION_PATTERN
- ADMIN_SERIAL_DISTRIBUTION
- ADMIN_SERIAL_STREAM
- ADMIN_SERIAL_SUBSCRIPTION

To receive copies of serials:

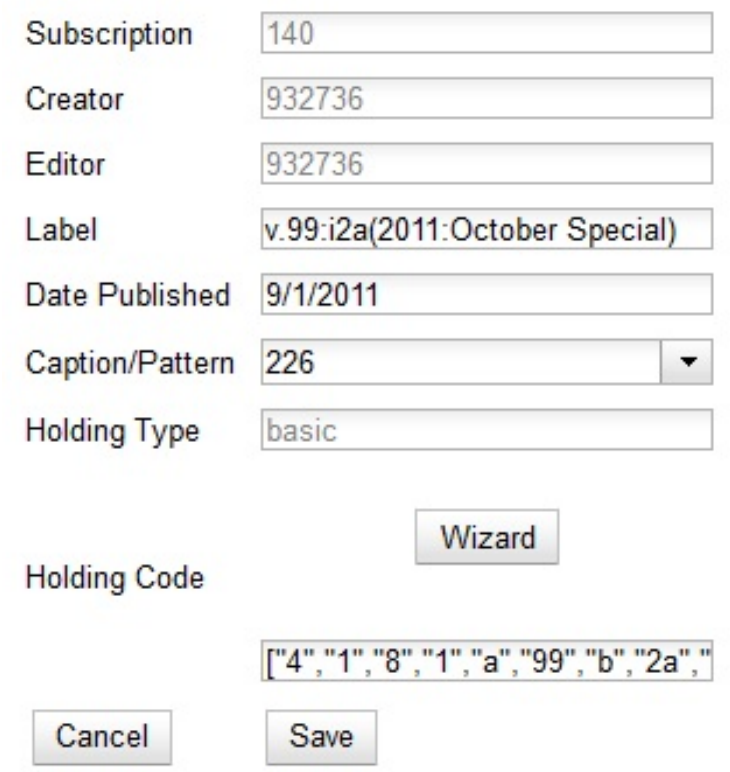
- RECEIVE_SERIAL
- CREATE_VOLUME

You only need the CREATE_VOLUME permission if you are barcoding items and creating new call numbers per issue.

Creating a Special Issue to Receive

Sometimes you may have to create a special issue to receive that occur outside of the standard publication pattern. To do this you need to do the following:

1. Create an issuance for the special edition



2. Click on the name of your special issue in the list of issuances

3. Click the New Items button

4. Enter the appropriate information

5. The item is now ready to receive. If you complete the Date Received field and change the status to received then it will receive the issue but it won't create the associated copy record whereas if you leave it blank and receive the item through the Serials Control View or Batch Receive function you can create the Copy Record at that time.

Part IV. Administration

This part of the documentation is intended for Evergreen administrators and requires root access to your Evergreen server(s) and administrator access to the Evergreen staff client. It deals with maintaining servers, installation, upgrading, and configuring both system wide and local library settings. Some sections require understanding of `Linux` system administration while others require an understanding of your system hierarchy of locations and users. Many procedures explained in the following chapters are accomplished with `Linux` commands run from the terminal without a Graphical User Interface (GUI).

In order to accomplish some of the tasks, prerequisite knowledge or experience will be required and you may need to consult system administration documentation for your specific `Linux` distribution if you have limited `Linux` system experience. A vast ammount of free resources can be found on the on the web for various experinece levels. You might also consider consulting PostgreSQL and Apache documentation for a greater understanding of the software stack on which Evergreen is built.

Chapter 9. System Requirements and Hardware Configurations

Evergreen is extremely scalable and can serve the need of a large range of libraries. The specific requirements and configuration of your system should be determined based on your specific needs of your organization or consortium.

Server Minimum Requirements

The following are the base requirements setting Evergreen up on a test server:

- An available desktop, server or virtual image
- 1GB RAM, or more if your server also runs a graphical desktop
- Linux Operating System
- Ports 80 and 443 should be opened in your firewall for TCP connections to allow OPAC and staff client connections to the Evergreen server.

Debian and Ubuntu are the most widely used Linux distributions for installing Evergreen and most development takes place on Debian based systems. If you are new to Linux, it is strongly recommended that you install Evergreen on the latest stable server edition of Debian (http://www.debian.org/) or Ubuntu 10.04 Server(http://www.ubuntu.com/) since the installation instructions have been tested on these distributions. Debian and Ubuntu are free distributions of Linux.

Server Hardware Configurations and Clustering

The hardware requirements for running a functional Evergreen server are minimal. It is also possible to scale up your evergreen configuration to be spread your Evergreen resources and services over several or even many servers in a clustered approach for the purpose of system redundancy, load balancing and downtime reduction. This allows very large consortia to share one Evergreen system with hundreds of libraries with millions of records and millions of users, making the scalability of Evergreen almost infinite.

Here are some example scenarios for networked server configurations:

- A small library library with 1 location, under 25,000 items and a few thousand users could easily run Evergreen on a single server (1 machine).
- A college or university with 1 million items and 20,000 users could run an Evergreen system using several servers balancing the load on their system by spreading services over multiple servers. It should host their PostgreSQL database on a separate server. They could also cluster the Evergreen services strategically to minimize or eliminate any necessary downtown when upgrading Evergreen or other server software. Moreover, system redundancy will reduce the chance of unplanned catastrophic downtime caused by system failure since Evergreen will be running over several machines.

- A large library consortium with several public library systems and/or academic libraries with millions of users and items could run an Evergreen system over many servers with clusters for Evergreen services as well as a cluster for the Postgresql Database.

The key to Evergreen scalability is in the OpenSRF configuration files `/openils/conf/opensrf.xml` and `/openils/conf/opensrf_core.xml`. By configuring these files, an administrator could cluster evergreen services over multiple hosts, change the host running a specific service or change the host of the PostgreSQL database.

The default configuration of Evergreen in the installation instructions assumes a single `localhost` server setup. For more complex multi-server clustered configurations, some server administration and database administration experience or knowledge will be required.

Staff Client Requirements

Staff terminals connect to the central database using the Evergreen staff client, available for download from The Evergreen download page. The staff client must be installed on each staff workstation and requires at minimum:

- `Windows` (XP, Vista, or 7), `Mac OS X`, or `Linux` operating system
- a reliable high speed Internet connection
- 512Mb of RAM
- The staff client uses the `TCP` protocal on ports `80` and `443` to communicate with the Evergreen server.

Barcode Scanners

Evergreen will work with virtually any barcode scanner – if it worked with your legacy system it should work on Evergreen.

Printers

Evergreen can use any printer configured for your terminal to print receipts, check-out slips, holds lists, etc. The single exception is spine label printing, which is still under development. Evergreen currently formats spine labels for output to a label roll printer. If you do not have a roll printer manual formatting may be required. For more on configuring receipt printers, see Printer Settings.

Chapter 10. Installing the Evergreen Server

Preamble: referenced user accounts

In subsequent simplesects, we will refer to a number of different accounts, as follows:

- Linux user accounts:
  - The **user** Linux account is the account that you use to log onto the Linux system as a regular user.
  - The **root** Linux account is an account that has system administrator privileges. On Debian and Fedora you can switch to this account from your **user** account by issuing the `su -` command and entering the password for the **root** account when prompted. On Ubuntu you can switch to this account from your **user** account using the `sudo su -` command and entering the password for your **user** account when prompted.
  - The **opensrf** Linux account is an account that you create when installing OpenSRF. You can switch to this account from the **root** account by issuing the `su - opensrf` command.
  - The **postgres** Linux account is created automatically when you install the PostgreSQL database server. You can switch to this account from the **root** account by issuing the `su - postgres` command.
- PostgreSQL user accounts:
  - The **evergreen** PostgreSQL account is a superuser account that you will create to connect to the PostgreSQL database server.
- Evergreen administrator account:
  - The **egadmin** Evergreen account is an administrator account for Evergreen that you will use to test connectivity and configure your Evergreen instance.

Preamble: Getting an Evergreen official release tarball

To download and extract the source for the current release of Evergreen, issue the following commands as the **user** Linux account:

```
wget -c http://evergreen-ils.org/downloads/Evergreen-ILS-2.1.1.tar.gz
tar xzf Evergreen-ILS-2.1.1.tar.gz
```

Preamble: Developer instructions

Skip this section if you are using an official release tarball downloaded from http://evergreen-ils.org/downloads

Developers working directly with the source code from the Git repository, rather than an official release tarball, must install some extra packages and perform one step before they can proceed with the `./configure` step.

As the **root** Linux account, install the following packages:

- autoconf
- automake
- libtool

As the **user** Linux account, issue the following command in the Evergreen source directory to generate the configure script and Makefiles:

```
./autogen.sh
```

After running `make install`, developers also need to install the Dojo Toolkit set of JavaScript libraries. The appropriate version of Dojo is included in Evergreen release tarballs. Developers should install the Dojo 1.3.3 version of Dojo by issuing the following commands as the **opensrf** Linux account:

```
wget http://download.dojotoolkit.org/release-1.3.3/dojo-release-1.3.3.tar.gz
tar -C /openils/var/web/js -xzf dojo-release-1.3.3.tar.gz
cp -r /openils/var/web/js/dojo-release-1.3.3/* /openils/var/web/js/dojo/.
```

Installing prerequisites

Evergreen has a number of prerequisite packages that must be installed before you can successfully configure, compile, and install Evergreen.

1. Begin by installing the most recent version of OpenSRF (2.0 or later). You can download OpenSRF releases from http://evergreen-ils.org/opensrf.php
2. On many distributions, it is necessary to install PostgreSQL 9 from external repositories.
   - On Debian Squeeze, open `/etc/apt/sources.list` in a text editor as the **root** Linux account and add the following line:

     ```
     deb http://backports.debian.org/debian-backports squeeze-backports main contrib
     ```

   - On Ubuntu Lucid, you can use a PPA (personal package archive), which are package sources hosted on Launchpad. The one most commonly used by Evergreen Community members is maintained by Martin Pitt, who also maintains the official PostgreSQL packages for Ubuntu. As the **root** Linux account, issue the following commands to add the PPA source:

     ```
     apt-get install python-software-properties
     add-apt-repository ppa:pitti/postgresql
     ```

   - Fedora 15 comes with PostgreSQL 9, so no additional steps are required.
3. On Debian and Ubuntu, run `aptitude update` as the **root** Linux account to retrieve the new packages from the backports repository.
4. Issue the following commands as the **root** Linux account to install prerequisites using the `Makefile.install` prerequisite installer, substituting `debian-squeeze`, `fedora15`, `ubuntu-lucid`, `centos`, or `rhel` for <osname> below:

   ```
   make -f Open-ILS/src/extras/Makefile.install <osname>
   ```

`centos` and `rhel` are less tested than the `debian`, `fedora`, and `ubuntu` options. Your patches and suggestions for improvement are welcome!

5. Add the libdbi-libdbd libraries to the system dynamic library path by issuing the following commands as the **root** Linux account:

   Debian / Ubuntu.

   ```
   echo "/usr/local/lib/dbd" > /etc/ld.so.conf.d/eg.conf
   ldconfig
   ```

   Fedora.

   ```
   echo "/usr/lib64/dbd" > /etc/ld.so.conf.d/eg.conf
   ldconfig
   ```

Configuration and compilation instructions

For the time being, we are still installing everything in the `/openils/` directory. From the Evergreen source directory, issue the following commands as the **user** Linux account to configure and build Evergreen:

```
./configure --prefix=/openils --sysconfdir=/openils/conf
make
```

Installation instructions

1. Once you have configured and compiled Evergreen, issue the following command as the **root** Linux account to install Evergreen, build the server portion of the staff client, and copy example configuration files to `/openils/conf`. Change the value of the `STAFF_CLIENT_STAMP_ID` variable to match the version of the staff client that you will use to connect to the Evergreen server.

   ```
   make STAFF_CLIENT_STAMP_ID=rel_2_1_1 install
   ```

2. The server portion of the staff client expects `http://hostname/xul/server` to resolve. Issue the following commands as the **root** Linux account to create a symbolic link pointing to the `server` subdirectory of the server portion of the staff client that we just built using the staff client ID *rel_name*:

   ```
   cd /openils/var/web/xul
   ln -sf rel_name/server server
   ```

Change ownership of the Evergreen files

All files in the `/openils/` directory and subdirectories must be owned by the `opensrf` user. Issue the following command as the **root** Linux account to change the ownership on the files:

```
chown -R opensrf:opensrf /openils
```

Create the oils_web.xml configuration file

Many administration interfaces, such as acquisitions, bookings, and various configuration screens, depend on the correct configuration of HTML templates. Copying the sample configuration file into place should work in most cases:

```
cp /openils/conf/oils_web.xml.example /openils/conf/oils_web.xml
```

Configure the Apache Web server

1. Use the example configuration files in `Open-ILS/examples/apache/` to configure your Web server for the Evergreen catalog, staff client, Web services, and administration interfaces. Issue the following commands as the **root** Linux account:

   Debian and Ubuntu.

   ```
   cp Open-ILS/examples/apache/eg.conf       /etc/apache2/sites-available/
   cp Open-ILS/examples/apache/eg_vhost.conf /etc/apache2/
   cp Open-ILS/examples/apache/startup.pl    /etc/apache2/
   # Now set up SSL
   mkdir /etc/apache2/ssl
   cd /etc/apache2/ssl
   ```

   Fedora.

   ```
   cp Open-ILS/examples/apache/eg.conf       /etc/httpd/sites-available/
   cp Open-ILS/examples/apache/eg_vhost.conf /etc/httpd/
   cp Open-ILS/examples/apache/startup.pl    /etc/httpd/
   # Now set up SSL
   mkdir /etc/httpd/ssl
   cd /etc/httpd/ssl
   ```

2. The `openssl` command cuts a new SSL key for your Apache server. For a production server, you should purchase a signed SSL certificate, but you can just use a self-signed certificate and accept the warnings in the staff client and browser during testing and development. Create an SSL key for the Apache server by issuing the following command as the **root** Linux account:

   ```
   openssl req -new -x509 -days 365 -nodes -out server.crt -keyout server.key
   ```

3. As the **root** Linux account, edit the `eg.conf` file that you copied into place.

   a. Replace `Allow from 10.0.0.0/8` with `Allow from all` (to enable access to the offline upload / execute interface from any workstation on any network - note that you must secure this for a production instance)

4. Change the user for the Apache server.

   - (Debian and Ubuntu): As the **root** Linux account, edit `/etc/apache2/envvars`. Change `export APACHE_RUN_USER=www-data` to `export APACHE_RUN_USER=opensrf`.

   - (Fedora): As the **root** Linux account , edit `/etc/httpd/conf/httpd.conf`. Change `User apache` to `User opensrf`.

5. Configure Apache with performance settings appropriate for Evergreen:

   - (Debian and Ubuntu): As the **root** Linux account, edit `/etc/apache2/apache2.conf`:

   - (Fedora): As the **root** Linux account, edit `/etc/httpd/conf/httpd.conf`:

     a. Change `KeepAliveTimeout` to `1`. Higher values reduce the chance of a request timing out unexpectedly, but increase the risk of using up all available Apache child processes.

     b. *Optional*: Change `MaxKeepAliveRequests` to `100`

c. Update the prefork configuration simplesect to suit your environment. The following settings apply to a busy system:

```
<IfModule mpm_prefork_module>
   StartServers           20
   MinSpareServers         5
   MaxSpareServers        15
   MaxClients            150
   MaxRequestsPerChild 10000
</IfModule>
```

6. (Debian and Ubuntu): As the **root** Linux account, enable the Evergreen site:

```
a2dissite default  # OPTIONAL: disable the default site (the "It Works" page)
a2ensite eg.conf
```

Configure OpenSRF for the Evergreen application

There are a number of example OpenSRF configuration files in `/openils/conf/` that you can use as a template for your Evergreen installation. Issue the following commands as the **opensrf** Linux account:

```
cp -b /openils/conf/opensrf_core.xml.example /openils/conf/opensrf_core.xml
cp -b /openils/conf/opensrf.xml.example /openils/conf/opensrf.xml
```

When you installed OpenSRF, you created four Jabber users on two separate domains and edited the `opensrf_core.xml` file accordingly. Please refer back to the OpenSRF README and, as the **opensrf** Linux account, edit the Evergreen version of the `opensrf_core.xml` file using the same Jabber users and domains as you used while installing and testing OpenSRF.

The `-b` flag tells the `cp` command to create a backup version of the destination file. The backup version of the destination file has a tilde (~) appended to the file name, so if you have forgotten the Jabber users and domains, you can retrieve the settings from the backup version of the files.

`eg_db_config.pl`, described in the following simplesect, sets the database connection information in `opensrf.xml` for you.

Creating the Evergreen database

By default, the `Makefile.install` prerequisite installer does not install the PostgreSQL 9.0 database server required by every Evergreen system; for production use, most libraries install the PostgreSQL database server on a dedicated machine. You can install the packages required by Debian, Ubuntu, or Fedora on the machine of your choice using the following commands as the **root** Linux account:

(Debian / Ubuntu) Installing PostgreSQL 9.0 server packages.

```
make -f Open-ILS/src/extras/Makefile.install install_pgsql_server_debs_90
```

(Fedora 15) Installing PostgreSQL 9.0 server packages.

```
make -f Open-ILS/src/extras/Makefile.install install_fedora_pgsql_server
```

For a standalone PostgreSQL server, install the following Perl modules as the **root** Linux account:

(Debian / Ubuntu) Installing additional Perl modules on a standalone PostgreSQL 9.0 server.

```
aptitude install gcc libxml-libxml-perl libxml-libxslt-perl
cpan Business::ISBN
cpan JSON::XS
cpan Library::CallNumber::LC
cpan MARC::Record
cpan MARC::File::XML
cpan UUID::Tiny
```

(Fedora 15) Installing additional Perl modules on a standalone PostgreSQL 9.0 server.

```
yum install gcc perl-XML-LibXML perl-XML-LibXSLT perl-Business-ISBN
cpan JSON::XS
cpan Library::CallNumber::LC
cpan MARC::Record
cpan MARC::File::XML
cpan UUID::Tiny
```

You need to create a PostgreSQL superuser to create and access the database. Issue the following command as the **postgres** Linux account to create a new PostgreSQL superuser named `evergreen`. When prompted, enter the new user's password:

```
createuser -s -P evergreen
```

Once you have created the **evergreen** PostgreSQL account, you also need to create the database and schema, and configure your configuration files to point at the database server. Issue the following command as the **root** Linux account from inside the Evergreen source directory, replacing <user>, <password>, <hostname>, <port>, and <dbname> with the appropriate values for your PostgreSQL database (where <user> and <password> are for the **evergreen** PostgreSQL account you just created), and replace <admin-user> and <admin-pass> with the values you want for the **egadmin** Evergreen administrator account:

```
perl Open-ILS/src/support-scripts/eg_db_config.pl --update-config \
       --service all --create-database --create-schema --create-offline \
       --user <user> --password <password> --hostname <hostname> --port <port> \
       --database <dbname> --admin-user <admin-user> --admin-pass <admin-pass>
```

This creates the database and schema and configures all of the services in your `/openils/conf/opensrf.xml` configuration file to point to that database. It also creates the configuration files required by the Evergreen `cgi-bin` administration scripts, and sets the user name and password for the **egadmin** Evergreen administrator account to your requested values.

Creating the database on a remote server

In a production instance of Evergreen, your PostgreSQL server should be installed on a dedicated server. To create the database in that case, you can either:

- Install the PostgreSQL contrib modules on the machine on which you are installing the Evergreen code, and use the --create-database option from that machine, or
- Copy the `Open-ILS/src/sql/Pg/create_database.sql` script to your PostgreSQL server and invoke it as the **postgres** Linux account:

  ```
  psql -vdb_name=<dbname> -vcontrib_dir=`pg_config --sharedir`/contrib -f create_database.sql
  ```

Then you can issue the `eg_db_config.pl` command as above *without* the `--create-database` argument to create your schema and configure your configuration files.

Starting Evergreen

1. As the **root** Linux account, start the `memcached` and `ejabberd` services (if they aren't already running):

```
/etc/init.d/ejabberd start
/etc/init.d/memcached start
```

2. As the **opensrf** Linux account, start Evergreen. The `-l` flag in the following command is only necessary if you want to force Evergreen to treat the hostname as `localhost`; if you configured `opensrf.xml` using the real hostname of your machine as returned by `perl -ENet::Domain 'print Net::Domain::hostfqdn() . "\n";'`, you should not use the `-l` flag.

   ```
   osrf_ctl.sh -l -a start_all
   ```

   - If you receive the error message `bash: osrf_ctl.sh: command not found`, then your environment variable `PATH` does not include the `/openils/bin` directory; this should have been set in the **opensrf** Linux account's `.bashrc` configuration file. To manually set the `PATH` variable, edit the configuration file `~/.bashrc` as the **opensrf** Linux account and add the following line:

     ```
     export PATH=$PATH:/openils/bin
     ```

3. As the **opensrf** Linux account, generate the Web files needed by the staff client and catalogue and update the organization unit proximity (you need to do this the first time you start Evergreen, and after that each time you change the library hierarchy in `config.cgi`):

   ```
   autogen.sh -u
   ```

4. As the **root** Linux account, restart the Apache Web server:

   ```
   /etc/init.d/apache2 restart
   ```

   If the Apache Web server was running when you started the OpenSRF services, you might not be able to successfully log in to the OPAC or staff client until the Apache Web server is restarted.

Testing connections to Evergreen

Once you have installed and started Evergreen, test your connection to Evergreen via `srfsh`. As the **opensrf** Linux account, issue the following commands to start `srfsh` and try to log onto the Evergreen server using the **egadmin** Evergreen administrator user name and password that you set using the `eg_db_config.pl` command:

```
/openils/bin/srfsh
srfsh% login <admin-user> <admin-pass>
```

You should see a result like:

```
Received Data: "250bf1518c7527a03249858687714376"
------------------------------------
Request Completed Successfully
Request Time in seconds: 0.045286
------------------------------------

Received Data: {
   "ilsevent":0,
   "textcode":"SUCCESS",
   "desc":" ",
   "pid":21616,
   "stacktrace":"oils_auth.c:304",
   "payload":{
      "authtoken":"e5f9827cc0f93b503a1cc66bee6bdd1a",
      "authtime":420
   }

}
```

```
-----------------------------------
Request Completed Successfully
Request Time in seconds: 1.336568
-----------------------------------
```

If this does not work, it’s time to do some troubleshooting.

- As the **opensrf** Linux acccount, run the `settings-tester.pl` script to see if it finds any system configuration problems. The script is found at `Open-ILS/src/support-scripts/settings-tester.pl` in the Evergreen source tree.
- Follow the steps in the troubleshooting guide.
- If you have faithfully followed the entire set of installation steps listed here, you are probably extremely close to a working system. Gather your configuration files and log files and contact the Evergreen development mailing list for assistance before making any drastic changes to your system configuration.

Getting help

Need help installing or using Evergreen? Join the mailing lists at http://evergreen-ils.org/listserv.php or contact us on the Freenode IRC network on the #evergreen channel.

Chapter 11. Upgrading Evergreen to 2.1

Abstract

This Chapter will explain the step-by-step process of upgrading Evergreen to 2.1, including steps to upgrade OpenSRF. Before upgrading, it is important to carefully plan an upgrade strategy to minimize system downtime and service interruptions. All of the steps in this chapter are to be completed from the command line.

Evergreen 2.1 has several software requirements:

- PostgreSQL: Version 9.0 is the minimum supported version of PostgreSQL.
- Linux: Evergreen 2.0 has been tested on Debian Squeeze (6.0) and Ubuntu Lucid Lynx (10.04). If you are running an older version of these distributions, you may want to upgrade before installing Evergreen 2.1. For instructions on upgrading these distributions, visit the Debian or Ubuntu websites.

In the following instructions, you are asked to perform certain steps as either the `root` or `opensrf` user.

- Debian: To become the `root` user, issue the **su** command and enter the password of the `root` user.
- Ubuntu: To become the `root` user, issue the **sudo su** command and enter the password of your current user.

To switch from the `root` user to a different user, issue the **su - [user]** command; for example, **su - opensrf**. Once you have become a non-root user, to become the `root` user again simply issue the exit command.

In the following instructions, `/path/to/OpenSRF/` represents the path to the OpenSRF source directory.

1. Stop Evergreen and back up your data:
   a. As `root`, stop the Apache web server.
   b. As the `opensrf` user, stop all Evergreen and OpenSRF services:

      ```
      osrf_ctl.sh -l -a stop_all
      ```

   c. Back up of the `/openils` directory.
   d. Back up the evergreen database.
2. Upgrade OpenSRF to the latest edition.

   Download and install the latest version of Opensrf from the OpenSRF download page.
3. As the `opensrf` user, download and extract Evergreen 2.1

   ```
   wget http://www.open-ils.org/downloads/Evergreen-ILS-2.1.1.tar.gz
   tar xzf Evergreen-ILS-2.1.1.tar.gz
   ```

For the latest edition of Evergreen 2.1, check the Evergreen download page at http://www.open-ils.org/downloads.php and adjust upgrading instructions accordingly.

4. As the `root` user, install the prerequisites:

```
cd /home/opensrf/Evergreen-ILS-2.1.1
```

On the next command, replace `[distribution]` with one of these values for your distribution of Debian or Ubuntu:

- `debian-squeeze` for `Debian Squeeze (6.0)`
- `ubuntu-lucid` for `Ubuntu Lucid Lynx (10.04)`

```
make -f Open-ILS/src/extras/Makefile.install [distribution]
```

5. As the `opensrf` user, configure and compile Evergreen:

```
cd /home/opensrf/Evergreen-ILS-2.1.1

./configure --prefix=/openils --sysconfdir=/openils/conf

make
```

6. As the `root` user, install Evergreen:

```
make STAFF_CLIENT_BUILD_ID=rel_2_1_1 install
```

7. As the *root* user, change all files to be owned by the *opensrf* user and group:

```
chown -R opensrf:opensrf /openils
```

8. As the `opensrf` user, update configuration files:

```
cd /home/opensrf/Evergreen-ILS-2.1.1
# and offline-config.pl for the offline staff client data uploader
perl Open-ILS/src/support-scripts/eg_db_config.pl \
  --create-offline --user evergreen --password evergreen \
  --hostname localhost --port 5432 --database evergreen
```

9. As the `opensrf` user, update server symlink in `/openils/var/web/xul/`:

```
cd /openils/var/web/xul/
rm server
ln -s rel_2_1_1/server
```

10. Change to the Evergreen installation directory:

```
cd /home/opensrf/Evergreen-ILS-2.1.1
```

11. Update the evergreen database:

It is recommended that you back up your Evergreen database in order to restore your data if anything goes wrong.

If you were running Evergreen 2.0.x on PostgreSQL 8.4, you will need to upgrade to PostgreSQL 9.0:

a. Install the PostgreSQL 9.0 server packages following the installation instructions. Pay close attention to the backports section.

b. Upgrade your existing PostgreSQL 8.4 cluster to a PostgreSQL 9.0 cluster by issuing the following commands as `root`:

   i. Blow away the default PostgreSQL 9.0 main cluster:

   ```
   pg_dropcluster 9.0 main
   ```

   ii. Upgrade your production PostgreSQL 8.4 main cluster

   ```
   pg_upgradecluster --stop 8.4 main
   ```

   iii. OPTIONAL: Blow away your old PostgreSQL 8.4 main cluster. If you don't do this, then you might need to update `opensrf.xml` with new port numbers (probably 5433)

   ```
   pg_dropcluster 8.4 main
   ```

c. Add the hstore PostgreSQL contrib module to your Evergreen database:

```
psql -U evergreen -h localhost -f /usr/share/postgresql/9.0/contrib/hstore.sql -d evergreen
```

12. Upgrade Evergreen Schema.

Pay attention to error output as you run these scripts. You should do additional troubleshooting and error reporting to the Evergreen Developer List if you encounter errors.

```
cd /home/opensrf/Evergreen-ILS-2.1.1
psql -U evergreen -h localhost -f Open-ILS/src/sql/Pg/2.0-2.1-upgrade-db.sql evergreen
psql -U evergreen -h localhost -f Open-ILS/src/sql/Pg/2.1.0-2.1.1-upgrade-db.sql evergreen
```

13. As the `opensrf` user, copy `/openils/conf/oils_web.xml.example` to `/openils/conf/oils_web.xml` . (If upgrading from 1.6.1.x, `oils_web.xml` should already exist.)

```
cp /openils/conf/oils_web.xml.example /openils/conf/oils_web.xml
```

14. Update `opensrf_core.xml` and `opensrf.xml` by copying the new example files (`/openils/conf/opensrf_core.xml.example` and `/openils/conf/opensrf.xml`).

```
cp /openils/conf/opensrf_core.xml.example /openils/conf/opensrf_core.xml
```

```
cp /openils/conf/opensrf.xml.example /openils/conf/opensrf.xml
```

Copying these configuration files will remove any customizations you have made to them. Remember to redo your customizations after copying them.

15. Update Apache files:

Copying these Apache configuration files will remove any customizations you have made to them. Remember to redo your customizations after copying them. For example, if you purchased an SSL certificate, you will need to edit `eg.conf` to point to the appropriate SSL certificate files.

a. Update `/etc/apache2/startup.pl` by copying the example from `Open-ILS/examples/apache/startup.pl`.

b. Update /etc/apache2/eg_vhost.conf by copying the example from Open-ILS/examples/apache/eg_vhost.conf.

c. Update `/etc/apache2/sites-available/eg.conf` by copying the example from `Open-ILS/ examples/apache/eg.conf`.

16. Update `opensrf.xml` with the database connection info:

    If you are happy with the default settings in opensrf.xml.example, then:

```
cp -b /openils/conf/opensrf.xml.example /openils/conf/opensrf.xml
perl Open-ILS/src/support-scripts/eg_db_config.pl --update-config --service all \
  --database evergreen --host localhost --user evergreen --password evergreen
```

    Otherwise, compare /openils/conf/opensrf.xml with /openils/conf/opensrf.xml and manually copy the new pieces into place in your existing opensrf.xml file

17. As of Evergreen 2.1.1, the Perl modules are installed in the normal location on the system. To avoid conflicting versions, get the old versions of Perl modules out of your PERL5LIB path. Issue the following command as the opensrf user:

```
mv /openils/lib/perl5 /openils/lib/perl5-2.0
```

Restart Evergreen and Test

1. As the `opensrf` user, start all Evergreen and OpenSRF services:

```
osrf_ctl.sh -l -a start_all
```

2. As the `opensrf` user, run **autogen** to refresh the static organizational data files:

```
cd /openils/bin
./autogen.sh -c /openils/conf/opensrf_core.xml -u
```

3. Start **srfsh** and try logging in using your Evergreen username and password:

```
/openils/bin/srfsh
srfsh% login username password
```

4. Start the Apache web server.

If you encounter errors, refer to the troubleshooting section of this documentation for tips on finding solutions and seeking further assistance from the Evergreen community.

Chapter 12. Server Operations and Maintenance

Abstract

This chapter deals with basic server operations such as starting and stopping Evergreen as well wall security, backing up and troubleshooting Evergreen.

Starting, Stopping and Restarting

Occasionally, you may need to restart Evergreen. It is imperative that you understand the basic commands to stop and start the Evergreen server. You can start and stop Evergreen from the command line of the server using the **osrf_ctl.sh** script located in the `openils/bin` directory.

The osrf_ctl.sh command must be run as the `opensrf` user.

To view help on **osrf_ctl.sh** and get all of its options, run:

```
osrf_ctl.sh -h
```

To start Evergreen, run:

```
osrf_ctl.sh -l -a start_all
```

The `-l` flag is used to indicate that Evergreen is configured to use `localhost` as the host. If you have configured `opensrf.xml` to use your real hostname, do not use the `-l` flag. The `-a` option is required and indicates the *action* of the command. In this case `start_all`.

If you receive the error message: osrf_ctl.sh: command not found, then your environment variable `PATH` does not include the `/openils/bin` directory. You can set it using the following command:

```
export PATH=$PATH:/openils/bin
```

If you receive the error message Can't locate OpenSRF/System.pm in @INC … BEGIN failed–compilation aborted, then your environment variable `PERL5LIB` does not include the `/openils/lib/perl5` directory. You can set it using the following command:

```
export PERL5LIB=$PERL5LIB:/openils/lib/perl5
```

It is also possible to start a specific service. For example:

```
osrf_ctl.sh -l -a start_router
```

will only start the `router` service.

If you decide to start each service individually, you need to start them in a specific order for Evergreen to start correctly. Run the commands in this exact order:

```
osrf_ctl.sh -l -a start_router
```

```
osrf_ctl.sh -l -a start_perl

osrf_ctl.sh -l -a start_c
```

After starting or restarting Evergreen, it is also necessary to restart the `Apache web server` for the OPAC to work correctly.

To stop Evergreen, run:

```
osrf_ctl.sh -l -a stop_all
```

As with starting, you can choose to stop services individually.

To restart Evergreen, run:

```
osrf_ctl.sh -l -a restart_all
```

Backing Up

Backing up your system files and data is a critical task for server and database administrators. Having a strategy for backing up and recovery could be the difference between a minor annoyance for users and a complete catastrophe.

Backing up the Evergreen Database

Most of the critical data for an Evergreen system – patrons, bibliographic records, holdings, transactions, bills – is stored in the PostgreSQL database. You can therefore use normal PostgreSQL backup procedures to backup this data. For example, the simplest method of backing up the Evergreen database is to use the **pg_dump** command to create a live backup of the database without having to interrupt any Evergreen services. Here is an example pg_dump command which will dump a local Evergreen database into a the file `evergreen_db.backup`:

```
pg_dump -U evergreen -h localhost -f evergreen_db.backup evergreen
```

To restore the backed up database into a new database, create a new database using the template0 database template and the UTF8 encoding, and run the **psql** command, specifying the new database as your target:

```
createdb -T template0 -E UTF8 -U evergreen -h localhost new_evergreen

psql -U evergreen -h localhost -f evergreen_db.backup new_evergreen
```

This method of backup is only suitable for small Evergreen instances. Larger sites should consider implementing continuous archiving (also known as “log shipping”) to provide more granular backups with lower system overhead. More information on backing up PostgreSQL databases can be found in the official PostgreSQL documentation.

Backing up Evergreen Files

When you deploy Evergreen, you will probably customize many aspects of your system including the system configuration files, Apache configuration files, OPAC and Staff Client. In order to protect your investment of time, you should carefully consider the best approach to backing up files.

There are a number of ways of tackling this problem. You could create a script that regularly creates a time-stamped tarball of all of these files and copies it to a remote server - but that would build up over time to hundreds of files.

You could use rsync to ensure that the files of interest are regularly updated on a remote server - but then you would lose track of the changes to the files, should you make a change that introduces a problem down the road.

Perhaps one of the best options is to use a version control system like Bazaar, git or Subversion to regularly push updates of the files you care about to a repository on a remote server. This gives you the advantage of quickly being able to run through the history of the changes you made, with a commenting system that reminds you why each change was made, combined with remote storage of the pertinent files in case of disaster on site. In addition, your team can create local copies of the repository and test their own changes in isolation from the production system. Using a version control system also helps to recover system customizations after an upgrade.

Full System Backup

A full system backup archives every file on the file system. Some basic methods require you to shut down most system processes; other methods can use mirrored RAID setups or SAN storage to take "snapshot" backups of your full system while the system continues to run. The subject of how to implement full system backups is beyond the scope of this documentation.

Security

As with an ILS and resource accessible from the world wide web careful consideration needs to be given to the security of your Evergreen servers and database. While it is impossible to cover all aspects of security, it is important to take several precautions when setting up production Evergreen site.

1. Change the Evergreen `admin` password and keep it secure. The default admin password is known by anyone who has installed Evergreen. It is not a secret and needs to be changed by the Administrator. It should also only be shared by those who need the highest level of access to your system.

2. Create strong passwords using a combination of numerical and alphabetical characters for all of the Administrative passwords including the `postgres` and `opensrf` users

3. Open ports in the firewall with caution - It is only necessary to open ports `80` and `443` for `TCP` connections to the Evergreen server from the OPAC and the staff client. It is critical for administrators to understand the concepts of network security and take precautions to minimize vulnerabilities.

4. Use permissions and permission groups wisely - it is important to understand the purpose of the permissions and to only give users the level of access that they require.

Managing Log Files

Evergreen comes with a sophisticated logging system, but it is important to manage the OpenSRF and Evergreen logs. This section will provide a couple of log management techniques and tools.

Using the `logrotate` Utility to Manage Log Size

Fortunately, this is not a new problem for `Unix` administrators, and there are a number of ways of keeping your logs under control. On `Debian` and `Ubuntu`, for example, the `logrotate` utility controls when old log files are compressed and a new log file is started. `logrotate` runs once a day and checks all log files that it knows about to see if a threshold of time or size has been reached and rotates the log files if a threshold condition has been met.

To teach `logrotate` to rotate Evergreen logs on a weekly basis, or if they are > 50MB in size, create a new file `/etc/logrotate.d/evergreen` with the following contents:

```
compress
/openils/var/log/*.log {
# keep the last 4 archived log files along with the current log file
 # log log.1.gz log.2.gz log.3.gz log.4.gz
 # and delete the oldest log file (what would have been log.5.gz)
rotate 5
# if the log file is > 50MB in size, rotate it immediately
size 50M
 # for those logs that don't grow fast, rotate them weekly anyway
  weekly
}
```

Changing Logging Level for Evergreen

Change the Log Levels in your config files. Changing the level of logging will help narrow down errors.

A high logging level is not wise to do in a production environment since it will produce vastly larger log files and thus reduce server performance.

Change logging levels by editing the configuration file `/openils/conf/opensrf_core.xml`

you will want to search for lines containing <loglevel>.

the default setting for loglevel is 3 which will log *errors*, *warnings* and *information*.

The next level is 4 which is for debugging and provides additional information helpful for the debugging process.

Thus, lines with:

```
<loglevel>3</loglevel>
```

Should be changed to:

```
<loglevel>4</loglevel>
```

to allow debugging level logging

Other logging levels include *0* for no logging, *1* for logging errors and *2* for logging warnings and errors.

Installing PostgreSQL from Source

Some `Linux` distributions, such as `Debian Etch (4.0)`, do not offer PostgreSQL version 8.2 as an installable package. Before you continue, examine the software dependencies listed in ??? to ensure that your Linux distribution supports the required version of PostgreSQL.

Some `Linux` distributions, such as `Debian Etch (4.0)`, do not offer PostgreSQL version 8.2 as an installable package. Before you continue, examine the software dependencies listed in ??? to ensure that your Linux distribution supports the required version of PostgreSQL.

1. Install the application stow on your system if it is not already installed. Issue the following command as the `root` user:

```
apt-get install stow
```

2. Download, compile, and install the latest release for PostgreSQL 8.2 (which was version `8.2.12` at the time of this writing). As the `root` user, follow these steps:

```
wget http://wwwmaster.postgresql.org/redir/198/h/source/v8.2.17/postgresql-8.2.17.tar.bz2
tar xzf postgresql-8.2.17.tar.gz
cd postgresql-8.2.17
./configure --with-perl --enable-integer-datetimes --with-openssl --prefix=/usr/local/stow/pgsql
make
make install
cd contrib
make
make install
cd xml2
make
make install
cd /usr/local/stow
stow pgsql
```

3. Create the new user `postgres` to run the PostgreSQL processes. As the `root` user, execute this command:

```
adduser postgres
```

4. Initialize the database directory and start up PostgreSQL. As the `root` user, follow these steps:

```
mkdir -p /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
initdb -D /usr/local/pgsql/data -E UNICODE --locale=C
pg_ctl -D /usr/local/pgsql/data -l /home/postgres/logfile start
```

If an error occurs during the final step above, review the path of the home directory for the `postgres` user. It may be `/var/lib/postresql` instead of `/home/postres`.

Configuring PostgreSQL

The values of several PostreSQL configuration parameters may be changed for enhanced performance. The following table lists the default values and some suggested updates for several useful parameters:

Table 12.1. Suggested configuration values

Parameter	Default	Suggested
default_statistics_target	10	100
work_mem	4Mb	128Mb
shared_buffers	8Mb	512Mb
effective_cache_size	128Mb	4Gb

Chapter 13. Migrating Data

Abstract

Migrating data into Evergreen can be one of the most daunting tasks for an administrator. This chapter will explain some procedures to help to migrate bibliographic records, copies and patrons into the Evergreen system. This chapter requires advanced ILS Administration experience, knowledge of Evergreen data structures, as well as knowledge of how to export data from your current system or access to data export files from your current system.

Migrating Bibliographic Records

One of the most important and challenging tasks is migrating your bibliographic records to a new system. The procedure may be different depending on the system from which you are migrating and the content of the marc records exported from the existing system. The procedures in this section deal with the process once the data from the existing system is exported into marc records. It does not cover exporting data from your existing non-Evergreen system.

Several tools for importing bibliographic records into Evergreen can be found in the Evergreen installation folder (`/home/opensrf/Evergreen-ILS-1.6.1.6/Open-ILS/src/extras/import/`) and are also available from the Evergreen repository (http://svn.open-ils.org/trac/ILS/browser/branches/rel_1_6_1/Open-ILS/src/extras/import).

Converting MARC records to Evergreen BRE JSON format

If you are starting with MARC records from your existing system or another source, use the marc2bre.pl script to create the JSON representation of a bibliographic record entry (hence bre) in Evergreen. `marc2bre.pl` can perform the following functions:

- Converts `MARC-8` encoded records to `UTF-8` encoding
- Converts `MARC21` to `MARCXML21`
- Select the unique record number field (common choices are '035' or '001'; check your records as you might be surprised how a supposedly unique field actually has duplicates, though marc2bre.pl will select a unique identifier for subsequent duplicates)
- Extracts certain pertinent fields for indexing and display purposes (along with the complete MARCXML21 record)
- Sets the ID number of the first record from this batch to be imported into the biblio.record_entry table (hint - run the following `SQL` to determine what this number should be to avoid conflicts:

```
psql -U postgres evergreen
 # SELECT MAX(id)+1 FROM biblio.record_entry;
```

- If you are processing multiple sets of MARC records with **marc2bre.pl** before loading the records into the database, you will need to keep track of the starting ID number for each subsequent batch of records that you

are importing. For example, if you are processing three files of MARC records with 10000 records each into a clean database, you would use `–startid 1`, `–startid 10001`, and `–startid 20001` parameters for each respective file.

- Ignore “trash” fields that you do not want to retain in Evergreen

- If you use `marc2bre.pl` to convert your MARC records from the `MARC-8` encoding to the UTF-8 encoding, it relies on the `MARC::Charset Perl` module to complete the conversion. When importing a large set of items, you can speed up the process by using a utility like `marc4j` or `marcdumper` to convert the records to `MARC21XML` and `UTF-8` before running them through **marc2bre.pl** with the `–marctype=XML` flag to tell **marc2bre.pl** that the records are already in `MARC21XML` format with the `UTF-8` encoding. If you take this approach, due to a current limitation of `MARC::File::XML` you have to do a horrible thing and ensure that there are no namespace prefixes in front of the element names. `marc2bre.pl` cannot parse the following example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<marc:collection xmlns:marc="http://www.loc.gov/MARC21/slim"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.loc.gov/MARC/slim
http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd">
  <marc:record>
    <marc:leader>00677nam a2200193 a 4500</marc:leader>
    <marc:controlfield tag="001">H01-0000844</marc:controlfield>
    <marc:controlfield tag="007">t </marc:controlfield>
    <marc:controlfield tag="008">060420s1950    xx            000 u fre d</marc:controlfield>
    <marc:datafield tag="040" ind1=" " ind2=" ">
      <marc:subfield code="a">CaOHCU</marc:subfield>
      <marc:subfield code="b">fre</marc:subfield>
    </marc:datafield>
...
;
```

But marc2bre.pl can parse the same example with the namespace prefixes removed:

```
<?xml version="1.0" encoding="UTF-8" ?>
<collection xmlns:marc="http://www.loc.gov/MARC21/slim"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.loc.gov/MARC/slim
http://www.loc.gov/standards/marcxml/schema/MARC21slim.xsd">
  <record>
    <leader>00677nam a2200193 a 4500</leader>
    <controlfield tag="001">H01-0000844</controlfield>
    <controlfield tag="007">t </controlfield>
    <controlfield tag="008">060420s1950    xx            000 u fre d</controlfield>
    <datafield tag="040" ind1=" " ind2=" ">
      <subfield code="a">CaOHCU</subfield>
      <subfield code="b">fre</subfield>
    </datafield>
...
;
```

Converting Records for Import into PostgreSQL

Once you have your records in Open-ILS `JSON` ingest format, you then need to use **pg_loader.pl** to convert these records into a set of `SQL` statements that you can use to load the records into PostgreSQL. The `–order` and `–autoprimary` command line options (bre, mrd, mfr, etc) map to class IDs defined in `/openils/conf/fm_IDL.xml`.

Adding Metarecords to the Database

One you have loaded the records into PostgreSQL, you can create metarecord entries in the metabib.metarecord table by running the following SQL:

```
psql evergreen
# \i /home/opensrf/Evergreen-ILS-1.6*/src/extras/import/quick_metarecord_map.sql
```

Metarecords are required to place holds on items, among other actions.

Migrating Bibliographic Records Using the ESI Migration Tools

The following procedure explains how to migrate bibliographic records from marc records into Evergreen. This is a general guide and will need to be adjusted for your specific environment. It does not cover exporting records from specific proprietary ILS systems. For assistance with exporting records from your current system please refer to the manuals for your system or you might try to ask for help from the Evergreen community.

1. Download the Evergreen migration utilities from the git repository.

   Use the command **git clone git://git.esilibrary.com/git/migration-tools.git** to clone the migration tools.

   Install the migration tools:

   ```
   cd migration-tools/Equinox-Migration
   perl Makefile.PL
   make
   make test
   make install
   ```

2. Add environmental variables for migration and import tools. These paths must point to:

   - the import perl scripts bundled with Evergreen
   - the folder where you extracted the migration tools
   - the location of the Equinox-Migration perl modules
   - the location of the Evergreen perl modules (e.g. perl5)

   ```
   export PATH=[path to Evergreen]/Open-ILS/src/extras/import: \
   /[path to migration-tools]/migration-tools:$PATH:.
   export PERL5LIB=/openils/lib/perl5: \
   /[path to migration-tools/Equinox-Migration/lib
   ```

3. Dump marc records into MARCXML using `yaz-marcdump`

```
echo '<?xml version="1.0" encoding="UTF-8" ?>' > imported_marc_records.xml
yaz-marcdump -f MARC-8 -t UTF-8 -o marcxml imported_marc_records.mrc >> imported_marc_records.xml
```

4. Test validity of XML file using `xmllint`

```
xmllint --noout imported_marc_records.xml 2> marc.xml.err
```

5. Clean up the marc xml file using the `marc_cleanup` utility:

```
marc_cleanup --marcfile=imported_marc_records.xml --fullauto [--renumber-from #] -ot 001
```

   The `--renumber-from` is required if you have bibliographic records already in your system. Use this to set the starting id number higher then the last id in the biblio.record_entry table. The marc_cleanup command will generate a file called `clean.marc.xml`

6. Create a fingerprinter file using the `fingerprinter` utility:

```
fingerprinter -o incumbent.fp -x incumbent.ex clean.marc.xml
```

   `fingerprinter` is used for deduplification of the incumbent records. The `-o` option specifies the output file and the `-x` option is used to specify the error output file.

7. Create a fingerprinter file for existing Evergreen bibliographic records using the `fingerprinter` utility if you have existing bibliographic records in your system previously imported:

```
fingerprinter -o production.fp -x production.fp.ex --marctype=MARC21 existing_marc_records.mrc \
--tag=901 --subfield=c
```

   `fingerprinter` is used for deduplification of the incumbant records.

8. Create a merged fingerprint file removing duplicate records.

```
cat cat production.fp incumbent.fp | sort -r > dedupe.fp
match_fingerprints [-t start id] -o records.merge dedupe.fp
```

9. Create a new import XML file using the `extract_loadset` utility

```
extract_loadset -l 1 -i clean.marc.xml -o merged.xml records.merge
```

10. Extract all of the currently used TCN's an generate the .bre and .ingest files to prepare for the bibliographic record load.

```
psql -U evergreen -c "select tcn_value from biblio.record_entry where not deleted" \
| perl -npe 's/^\s+//;' > used_tcns
marc2bre.pl --idfield 903 [--startid=#] --marctype=XML -f final.xml \
--used_tcn_file=used_tcns > evergreen_bre_import_file.bre
```

The option `--startid` needs to match the start id used in earlier steps and must be higher than largest id value in the biblio.record_entry table. the option `--idfield` should match the marc datafield used to store your records ids.

11. Ingest the bibliographic records into the Evergreen database.

```
parallel_pg_loader.pl \
-or bre \
-or mrd \
-or mfr \
-or mtfe \
-or mafe \
-or msfe \
-or mkfe \
-or msefe \
-a mrd \
-a mfr \
-a mtfe \
-a mafe \
-a msfe \
-a mkfe \
-a msefe evergreen_bre_import_file.bre > bibrecords.sql
```

12. Load the records using psql and the sql scripts generated from the previous step.

```
psql -U evergreen -h localhost -d evergreen -f bibrecords.sql
psql -U evergreen < ~/Ever*/Open-ILS/src/extras/import/quick_metarecord_map.sql > log.create_metabib
```

13. Extract holdings from marc records for importing copies into Evergreen using the `extract_holdings` utility.

```
extract_holdings --marcfile=clean.marc.xml --holding 999 --copyid 999i --map holdings.map
```

This command would extract holdings based on the 949 datafield in the marc records. The copy id is generated from the subfile i in the 999 datafield. You may need to adjust these options based on the field used for holdings informatiom in your marc records.

The `map` option `holdings.map` refers to a file to be used for mapping subfields to the holdings data you would like extracted. Here is an example based on mapping holdings data to the 999 data field:

```
callnum 999 a
barcode 999 i
location 999 l
owning_lib 999 m
circ_modifier 999 t
```

Running the extract holdings script should produce an sql script `HOLDINGS.pg` similar to:

```
BEGIN;

egid, hseq, l_callnum, l_barcode, l_location, l_owning_lib, l_circ_modifier,
40      0       HD3616.K853 U54 1997    30731100751928  STACKS  FENNELL BOOK
41      1       HV6548.C3 S984 1998     30731100826613  STACKS  FENNELL BOOK
41      2       HV6548.C3 S984 1998     30731100804958  STACKS  BRANTFORD       BOOK
...
```

Edit the holdings.pg sql script like so:

```
BEGIN;

TRUNCATE TABLE staging_items;

INSERT INTO staging_items (egid, hseq, l_callnum, l_barcode, l_location,
l_owning_lib, l_circ_modifier) FROM stdin;
40      0       HD3616.K853 U54 1997    30731100751928  STACKS  FENNELL BOOK
41      1       HV6548.C3 S984 1998     30731100826613  STACKS  FENNELL BOOK
41      2       HV6548.C3 S984 1998     30731100804958  STACKS  BRANTFORD       BOOK
\.

COMMIT;
```

This file can be used for importing holdings into Evergreen. the egid is a critical column. It is used to link the volume and copy to the bibliographic record. Please refer to for the steps to import your holdings into Evergreen.

Adding Copies to Bibliographic Records

Before bibliographic records can be found in an OPAC search copies will need to be created. It is very important to understand how various tables related to each other in regards to holdings maintenance.

The following procedure will guide you through the process of populating Evergreen with volumes and copies. This is a very simple example. The SQL queries may need to be adjusted for the specific data in your holdings.

1. Create a staging_items staging table to hold the holdings data:

```
CREATE TABLE staging_items (
        l_callnum text, -- call number label
 hseq int,
        egid  int,  -- biblio.record_entry_id
        createdate      date,
        l_location        text,
        l_barcode         text,
        l_circ_modifier   text,
        l_owning_lib      text  -- actor.org_unit.shortname
);
```

2. Import the items using the HOLDINGS.pg SQL script created using the extract_holdings utility.

```
psql -U evergreen -f HOLDINGS.pg evergreen
```

the file `HOLDINGS.pg` and/or the COPY query may need to be adjusted for your particular circumstances.

3. Generate shelving locations from your staging table.

```
INSERT INTO asset.copy_location (name, owning_lib)
SELECT  DISTINCT l.location, ou.id
FROM  staging_items l
       JOIN actor.org_unit ou ON (l.owning_lib = ou.shortname);
```

4. Generate circulation modifiers from your staging table.

```
INSERT INTO config.circ_modifier (code, name, description, sip2_media_type, magnetic_media)
        SELECT  DISTINCT l_circ_modifier AS code,
          l_circ_modifier AS name,
          LOWER(l_circ_modifier) AS description,
          '001' AS sip2_media_type,
          FALSE AS magnetic_media
          FROM  staging_items
          WHERE l_circ_modifier NOT IN (SELECT code FROM config.circ_modifier);
```

5. Generate call numbers from your staging table:

```
INSERT INTO asset.call_number (creator,editor,record,label,owning_lib)
SELECT  DISTINCT 1, 1, egid, l.callnum, ou.id
FROM  staging.staging_items l
JOIN actor.org_unit ou ON (l.owning_lib = ou.shortname);
```

6. Generate copies from your staging table:

```
INSERT INTO asset.copy (
circ_lib, creator, editor, create_date, barcode,
STATUS, location, loan_duration, fine_level, circ_modifier, deposit, ref, call_number)

SELECT  DISTINCT ou.id AS circ_lib,
 1 AS creator,
 1 AS editor,
 l.l_createdate AS create_date,
 l.l_barcode AS barcode,
 0 AS STATUS,
 cl.id AS location,
 2 AS loan_duration,
 2 AS fine_level,
 l.l_circ_modifier AS circ_modifier,
 FALSE AS deposit,
 CASE
 WHEN l.l_circ_modifier = 'REFERENCE' THEN TRUE
 ELSE FALSE
 END AS ref,
 cn.id AS call_number
 FROM  staging_items l
  JOIN actor.org_unit ou
   ON (l.l_owning_lib = ou.shortname)
  JOIN asset.copy_location cl
   ON (ou.id = cl.owning_lib AND l.l_location = cl.name)
  JOIN metabib.real_full_rec m
   ON (m.record = l.egid)
  JOIN asset.call_number cn
   ON (ou.id = cn.owning_lib
   AND m.record = cn.record
   AND l.l_callnum = cn.label)
```

You should now have copies in your Evergreen database and should be able to search and find the bibliographic records with attached copies.

Migrating Patron Data

This section will explain the task of migrating your patron data from comma delimited files into Evergreen. It does not deal with the process of exporting from the non-Evergreen system since this process may vary depending on where you are extracting your patron records. Patron could come from an ILS or it could come from a student database in the case of academic records.

When importing records into Evergreen you will need to populate 3 tables in your Evergreen database:

- actor.usr - The main table for user data
- actor.card - Stores the barcode for users; Users can have more than 1 card but only 1 can be active at a given time;
- actor.usr_address - Used for storing address information; A user can have more than one address.

Before following the procedures below to import patron data into Evergreen, it is a good idea to examine the fields in these tables in order to decide on a strategy for data to include in your import. It is important to understand the data types and constraints on each field.

1. Export the patron data from your existing ILS or from another source into a comma delimited file. The comma delimited file used for importing the records should use Unicode (UTF8) character encoding.

2. Create a staging table. A staging table will allow you to tweak the data before importing. Here is an example sql statement:

```
CREATE TABLE students (
  student_id int, barcode text, last_name text, first_name text, email text, address_type text, street1 text, st
 city text, province text, country text, postal_code text, phone text, profile int DEFAULT 2,
 ident_type int, home_ou int, claims_returned_count int DEFAULT 0, usrname text,
 net_access_level int DEFAULT 2, password text
);
```

   Note the DEFAULT variables. These allow you to set default for your library or to populate required fields if you data allows NULL values where fields are required in Evergreen.

   The data field profile in the above SQL script refers to the user group and should be an integer referencing the id field in permission.grp_tree. Setting this value will effect the permissions for the user. See the values in permission.grp_tree for possibilities.

   ident_type is the identification type used for identiying users. This is a integer value referencing config.identification_type and should match the id values of that table. The default values are 1 for Drivers License, 2 for SSN or 3 for other.

   home_ou is the home organizational unit for the user. This value needs to match the corresponding id in the actor.org_unit table.

3. Copy records into staging table from a comma delimited file.

```
COPY students (student_id, last_name, first_name, email, address_type, street1, street2, city, province, country
 FROM '/home/opensrf/patrons.csv'
  WITH CSV HEADER;
```

The above script wil vary depending on the format of your patron load file (`patrons.csv`). You may want to review PostgreSQL documentation

4. Formatting of some fields to fit Evergreen filed formatting may be required. Here is an example of sql to adjust phone numbers in the staging table to fit the evergreen field:

```
UPDATE students phone = replace(replace(replace(rpad(substring(phone from 1 for 9), 10, '-') ||
substring(phone from 10), '(', ''), ')', ''), ' ', '-');
```

Data “massaging” will be required to fit formats used in Evergreen.

5. Insert records from the staging table into the actor.usr Evergreen table:

```
 INSERT INTO actor.usr (
 profile, usrname, email, passwd, ident_type, ident_value, first_given_name,
 family_name, day_phone, home_ou, claims_returned_count, net_access_level)
 SELECT profile, students.usrname, email, password, ident_type, student_id,
 first_name, last_name, phone, home_ou, claims_returned_count, net_access_level
 FROM students;
```

6. insert records into actor.card from actor.usr .

```
INSERT INTO actor.card (usr, barcode)
 SELECT actor.usr.id, students.barcode
 FROM students
  INNER JOIN actor.usr
   ON students.usrname = actor.usr.usrname;
```

This assumes a one to one card patron relationship. If your patron data import has multiple cards assigned to one patron more complex import scripts may be required which look for inactive or active flags.

7. Update actor.usr.card field with actor.card.id to associate active card with the user:

```
UPDATE actor.usr
 SET card = actor.card.id
 FROM actor.card
 WHERE actor.card.usr = actor.usr.id;
```

8. Insert records into actor.usr_address to add address information for users:

```
INSERT INTO actor.usr_address (usr, street1, street2, city, state, country, post_code)
 SELECT actor.usr.id, students.street1, students.street2, students.city, students.province,
 students.country, students.postal_code
 FROM students
 INNER JOIN actor.usr ON students.usrname = actor.usr.usrname;
```

9. update actor.usr.address with address id from address table.

```
UPDATE actor.usr
 SET mailing_address = actor.usr_address.id, billing_address = actor.usr_address.id
 FROM actor.usr_address
 WHERE actor.usr.id = actor.usr_address.usr;
```

This assumes 1 address per patron. More complex scenarios may require more sophisticated SQL.

Creating an sql Script for Importing Patrons

The procedure for importing patron can be automated with the help of an sql script. Follow these steps to create an import script:

1. Create an new file and name it `import.sql`

2. Edit the file to look similar to this:

```
BEGIN;

-- Create staging table.
CREATE TABLE students (
 student_id int, barcode text, last_name text, first_name text, email text, address_type text, street1 text, str
 city text, province text, country text, postal_code text, phone text, profile int,
 ident_type int, home_ou int, claims_returned_count int DEFAULT 0, usrname text,
 net_access_level int DEFAULT 2, password text
);

--Copy records from your import text file
COPY students (student_id, last_name, first_name, email, address_type, street1, street2, city, province, country
 FROM '/home/opensrf/patrons.csv'
  WITH CSV HEADER;


--Insert records from the staging table into the actor.usr table.
INSERT INTO actor.usr (
 profile, usrname, email, passwd, ident_type, ident_value, first_given_name, family_name,
 day_phone, home_ou, claims_returned_count, net_access_level)
 SELECT profile, students.usrname, email, password, ident_type, student_id, first_name,
 last_name, phone, home_ou, claims_returned_count, net_access_level FROM students;

--Insert records from the staging table into the actor.usr table.
INSERT INTO actor.card (usr, barcode)
 SELECT actor.usr.id, students.barcode
 FROM students
  INNER JOIN actor.usr
   ON students.usrname = actor.usr.usrname;

--Update actor.usr.card field with actor.card.id to associate active card with the user:
UPDATE actor.usr
 SET card = actor.card.id
 FROM actor.card
 WHERE actor.card.usr = actor.usr.id;

--INSERT records INTO actor.usr_address from staging table.
INSERT INTO actor.usr_address (usr, street1, street2, city, state, country, post_code)
 SELECT actor.usr.id, students.street1, students.street2, students.city, students.province,
 students.country, students.postal_code
 FROM students
 INNER JOIN actor.usr ON students.usrname = actor.usr.usrname;


--Update actor.usr mailing address with id from actor.usr_address table.:
UPDATE actor.usr
 SET mailing_address = actor.usr_address.id, billing_address = actor.usr_address.id
 FROM actor.usr_address
 WHERE actor.usr.id = actor.usr_address.usr;

COMMIT;
```

Placing the sql statements between `BEGIN;` and `COMMIT;` creates a transaction block so that if any sql statements fail, the entire process is canceled and the database is rolled back to its original state. Lines beginning with `--` are comments to let you you what each sql statement is doing and are not processed.

Batch Updating Patron Data

For academic libraries, doing batch updates to add new patrons to the Evergreen database is a critical task. The above procedures and import script can be easily adapted to create an update script for importing new patrons from external databases. If the data import file contains only new patrons, then, the above procedures will work well to insert those patrons. However, if the data load contains all patrons, a second staging table and a procedure to remove existing patrons from that second staging table may be required before importing the new patrons. Moreover, additional steps to update address information and perhaps delete inactive patrons may also be desired depending on the requirements of the institution.

After developing the scripts to import and update patrons have been created, another important task for library staff is to develop an import strategy and schedule which suits the needs of the library. This could be determined by registration dates of your institution in the case of academic libraries. It is important to balance the convenience of patron loads and the cost of processing these loads vs staff adding patrons manually.

Restoring your Evergreen Database to an Empty State

If you've done a test import of records and you want to quickly get Evergreen back to a pristine state, you can create a clean Evergreen database schema by performing the following:

1. **`cd ILS/Open-ILS/src/sql/Pg/`**
2. Rebuild the database schema:

   `./build-db.sh [db-hostname> [db-port] [db-name] [db-user] [db-password] [db-version]`

   This will remove all of your data from the database and restore the default values.

Exporting Bibliographic Records into MARC files

The following procedure explains how to export Evergreen bibliographic records into MARC files using the **marc_export** support script. All steps should be performed by the `opensrf` user from your Evergreen server.

Processing time for exporting records depends on several factors such as the number of records you are exporting. It is recommended that you divide the export ID files (`records.txt`) into a manageable number of records if you are exporting a large number of records.

1. Create a text file list of the Bibliographic record IDs you would like to export from Evergreen. One way to do this is using SQL:

```
SELECT DISTINCT bre.id FROM biblio.record_entry AS bre
 JOIN asset.call_number AS acn ON acn.record = bre.id
 WHERE bre.deleted='false' and owning_lib=101 \g /home/opensrf/records.txt;
```

This query creates a file called `records.txt` containing a column of distinct IDs of items owned by the organizational unit with the id `101`.

2. Navigate to the support-scripts folder

```
cd /home/opensrf/Evergreen-ILS*/Open-ILS/src/support-scripts/
```

3. Run **marc_export**, using the ID file you created in step 1 to define which files to export.

```
cat /home/opensrf/records.txt | ./marc_export -i -c /openils/conf/opensrf_core.xml \
-x /openils/conf/fm_IDL.xml -f XML --timeout 5 > exported_files.xml
```

The example above exports the records into MARCXML format.

For help or for more options when running **marc_export**, run **marc_export** with the `-h` option:

```
./marc_export -h
```

Importing Authority Records

The following procedure explains how to export Evergreen bibliographic records into MARC files using the marc_export support script. All steps should be performed by the `opensrf` user from your Evergreen server.

Importing Authority Records from Command Line

The major advantages of the command line approach are its speed and its convenience for system administrators who can perform bulk loads of authority records in a controlled environment.

1. Run **marc2are.pl** against the authority records, specifying the user name, password, MARC type (USMARC or XML). Use `STDOUT` redirection to either pipe the output directly into the next command or into an output file for inspection. For example, to process a file with authority records in MARCXML format named `auth_small.xml` using the default user name and password, and directing the output into a file named `auth.are`:

```
cd Open-ILS/src/extras/import/
perl marc2are.pl --user admin --pass open-ils --marctype XML auth_small.xml > auth.are
```

The MARC type will default to USMARC if the `--marctype` option is not specified.

2. Run **pg_loader.pl** to generate the SQL necessary for importing the authority records into your system. To save time for very large batches of records, you could simply pipe the output of **marc2are.pl** directly into **pg_loader.pl**.

```
cd Open-ILS/src/extras/import/
 perl pg_loader.pl --auto are --order are auth.are > auth_load.sql
```

3. Load the authority records from the SQL file that you generated in the last step into your Evergreen database using the psql tool. Assuming the default user name, host name, and database name for an Evergreen instance, that command looks like:

```
psql -U evergreen -h localhost -d evergreen -f auth_load.sql
```

Importing authority records using the MARC Batch Import/Export interface from the Staff Client

Good for loading batches of up to 5,000 records (roughly) at a time, the major advantages to importing authority records using the MARC Batch Import/Export interface are that it does not require command-line or direct database access – good for both security in that it minimizes the number of people who need this access and for spreading the effort around to others in the library – and it does most of the work (for example, figuring out whether the batch of records is in XML or USMARC format) for you.

To import a set of MARC authority records from the MARC Batch Import/Export interface:

1. From the Evergreen staff client, select Cataloging → MARC Batch Import/Export. The Evergreen MARC File Upload screen opens, with Import Records as the highlighted tab.

2. From the Bibliographic records drop-down menu, select Authority records.

3. Enter a name for the queue (batch import job) in the Create a new upload queue field.

4. Select the Auto-Import Non-Colliding Records checkbox.

5. Click the Browse… button to select the file of MARC authorities to import.

6. Click the Upload button to begin importing the records. The screen displays Uploading… Processing… to show that the records are being transferred to the server, then displays a progress bar to show the actual import progress. When the staff client displays the progress bar, you can disconnect your staff client safely. Very large batches of records might time out at this stage.

7. Once the import is finished, the staff client displays the results of the import process. You can manually display the import progress by selecting the Inspect Queue tab of the MARC Batch Import/Export interface and selecting the queue name. By default, the staff client does not display records that were imported successfully; it only shows records that conflicted with existing entries in the database. The screen shows the overall status of the import process in the top right-hand corner, with the Total and Imported number of records for the queue.

Chapter 14. Administration Functions in the Acquisitions Module

Abstract

Currency Types

Currency types can be created and applied to funds in the administrative module. When a fund is applied to a copy or line item for purchase, the item will be purchased in the currency associated with that fund.

Create a currency type

1. To create a new currency type, click Admin # Server Administration # Acquisitions# Currency types.
2. Enter the currency code. No limits exist on the number of characters that can be entered in this field.
3. Enter the name of the currency type in Currency Label field. No limits exist on the number of characters that can be entered in this field.
4. Click Save.

Edit a currency type

1. To edit a currency type, click your cursor in the row that you want to edit. The row will turn blue.
2. Double-click. The pop-up box will appear, and you can edit the fields.
3. After making changes, click Save.

From the currency types interface, you can delete currencies that have never been applied to funds or used to make purchases.

Exchange Rates

Exchange rates define the rate of exchange between currencies. Evergreen will automatically calculate exchange rates for purchases. Evergreen assumes that the currency of the purchasing fund is identical to the currency of the provider, but it provides for two unique situations: If the currency of the fund that is used for the purchase is different from the currency of the provider as listed in the provider profile, then Evergreen will use the exchange rate to calculate the price of the item in the currency of the fund and debit the fund accordingly. When money is transferred between funds that use different currency types, Evergreen will automatically use the exchange rate to convert the money to the currency of the receiving fund. During such transfers, however, staff can override the automatic conversion by providing an explicit amount to credit to the receiving fund.

Create an exchange rate

1. To create a new exchange rate, click Admin # Server Administration # Acquisitions # Exchange Rates.

2. Click New Exchange Rate.
3. Enter the From Currency from the drop down menu populated by the currency types.
4. Enter the To Currency from the drop down menu populated by the currency types.
5. Enter the exchange Ratio.
6. Click Save.

Edit an Exchange Rate

Edit an exchange rate just as you would edit a currency type.

Funding Sources

Funding sources allow you to specify the sources that contribute monies to your fund(s). You can create as few or as many funding sources as you need.

Create a funding source

1. To create a new funding source, click Admin # Server Administration # Acquisitions # Funding Source.
2. Enter a funding source name. No limits exist on the number of characters that can be entered in this field.
3. Select an owner from the drop down menu. The owner indicates the organizational unit(s) whose staff can use this funding source. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).

The rule of parental inheritance applies to this list. For example, if a system is made the owner of a funding source, then users with appropriate permissions at the branches within the system could also use the funding source.

4. Create a code for the source. No limits exist on the number of characters that can be entered in this field.
5. Select a currency from the drop down menu. This menu is populated from the choices in the Currency Types interface.
6. Click Save.

Allocate Credits to Funding Sources

1. Apply a credit to this funding source.
2. Enter the amount of money that the funding source contributes to the organization. Funding sources are not tied to fiscal or calendar years, so you can continue to add money to the same funding source over multiple years, e.g. County Funding. Alternatively, you can name funding sources by year, e.g. County Funding 2010 and County Funding 2011, and apply credits each year to the matching source.
3. To apply a credit, click on the hyperlinked name of the funding source. The Funding Source Details will appear.

4. Click Apply credit.
5. Enter an amount to apply to this funding source.
6. Enter a note. This field is optional.
7. Click Apply.

Allocate credits to funds

If you have already set up your funds, then you can then click the Allocate to Fund button to apply credits from the funding sources to the funds. If you have not yet set up your funds, or you need to add a new one, you can allocate credits to funds from the funds interface. See section 1.2 for more information.

1. To allocate credits to funds, click Allocate to Fund.
2. Enter the amount that you want to allocate.
3. Enter a note. This field is optional.
4. Click Apply.

Track Debits and Credits

You can track credits to and allocations from each funding source. These amounts are updated when credits and allocations are made in the Funding Source Details. Access the Funding Source Details by clicking on the hyperlinked name of the Funding Source.

Fund Tags

You can apply tags to funds so that you can group funds for easy reporting. For example, you have three funds for children's materials: Children's Board Books, Children's DVDs, and Children's CDs. Assign a fund tag of "children's" to each fund. When you need to report on the amount that has been spent on all children's materials, you can run a report on the fund tag to find total expenditures on children's materials rather than reporting on each individual fund.

Create a Fund Tag

1. To create a fund tag, click Admin # Server Administration # Acquisitions # Fund Tags.
2. Click New Fund Tag. No limits exist on the number of characters that can be entered in this field.
3. Select a Fund Tag Owner from the drop down menu. The owner indicates the organizational unit(s) whose staff can use this fund tag. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).

The rule of parental inheritance applies to this list.

4. Enter a Fund Tag Name. No limits exist on the number of characters that can be entered in this field.

5. Click Save.

Funds

Funds allow you to allocate credits toward specific purchases. In the funds interface, you can create funds; allocate credits from funding sources to funds; transfer money between funds; and apply fund tags to funds. Funds are created for a specific year, either fiscal or calendar. These funds are owned by org units. At the top of the funds interface, you can set a contextual org unit and year. The drop down menu at the top of the screen enables you to focus on funds that are owned by specific organizational units during specific years.

Create a fund

1. To create a new fund, click Admin # Server Administration # Acquisitions # Funds.

2. Enter a name for the fund. No limits exist on the number of characters that can be entered in this field.

3. Create a code for the fund. No limits exist on the number of characters that can be entered in this field.

4. Enter a year for the fund. This can be a fiscal year or a calendar year. The format of the year is YYYY.

5. Select an org unit from the drop down menu. The org unit indicates the organizational units whose staff can use this fund. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).

The rule of parental inheritance applies to this list. See section

6. Select a currency type from the drop down menu. This menu is comprised of entries in the currency types menu. When a fund is applied to a line item or copy, the price of the item will be encumbered in the currency associated with the fund.

7. Click the Active box to activate this fund. You cannot make purchases from this fund if it is not active.

8. Enter a Balance Stop Percent. The balance stop percent prevents you from making purchases when only a specified amount of the fund remains. For example, if you want to spend 95 percent of your funds, leaving a five percent balance in the fund, then you would enter 95 in the field. When the fund reaches its balance stop percent, it will appear in red when you apply funds to copies.

9. Enter a Balance Warning Percent. The balance warning percent gives you a warning that the fund is low. You can specify any percent. For example, if you want to spend 90 percent of your funds and be warned when the fund has only 10 percent of its balance remaining, then enter 90 in the field. When the fund reaches its balance warning percent, it will appear in yellow when you apply funds to copies.

10. Check the Propagate box to propagate funds. When you propagate a fund, the ILS will create a new fund for the following fisca year with the same parameters as your current fund. All of the settings transfer except for the year and the amount of money in the fund. Propagation occurs during the fiscal year close-out operation.

11. Check the Rollover box if you want to roll over remaining funds into the same fund next year.

12. Click Save.

Allocate Credits from Funding Sources to Funds

Credits can be applied to funds from funding sources using the fund interface. The credits that you apply to the fund can be applied later to purchases.

1. To access funds, click Admin # Server Administration # Acquisitions # Funds.
2. Click the hyperlinked name of the fund.
3. To add a credit to the fund, click the Create Allocation tab.
4. Choose a Funding Source from the drop down menu.
5. Enter an amount that you want to apply to the fund from the funding source.
6. Enter a note. This field is optional.
7. Click Apply.

Transfer credits between funds

The credits that you allocate to funds can be transferred between funds if desired. In the following example, you can transfer $500.00 from the Young Adult Fiction fund to the Children's DVD fund.

1. To access funds, click Admin # Server Administration # Acquisitions # Funds.
2. Click the hyperlinked name of the originating fund.
3. The Fund Details screen appears. Click Transfer Money.
4. Enter the amount that you would like to transfer.
5. From the drop down menu, select the destination fund.
6. Add a note. This field is optional.
7. Click Transfer.

Track Balances and Expenditures

The Fund Details allows you to track the fund's balance, encumbrances, and amount spent. It also allows you to track allocations from the funding source(s), debits, and fund tags.

1. To access the fund details, click on the hyperlinked name of the fund that you created.
2. The Summary allows you to track the following:
    a. Balance – The balance is calculated by subtracting both items that have been invoiced and encumbrances from the total allocated to the fund.
    b. Total Allocated – This amount is the total amount allocated from the Funding Source.
    c. Spent Balance – This balance is calculated by subtracting only the items that have been invoiced from the total allocated to the fund. It does not include encumbrances.

d. Total Debits – The total debits are calculated by adding the cost of items that have been invoiced and encumbrances.

e. Total Spent – The total spent is calculated by adding the cost of items that have been invoiced. It does not include encumbrances.

f. Total Encumbered – The total encumbered is calculated by adding all encumbrances.

Edit a Fund

Edit a fund just as you would edit a currency type.

Perform Year End Closeout Operation

The Year End Closeout Operation allows you to deactivate funds for the current year and create analogous funds for the next year. It transfers encumbrances to the analogous funds, and it rolls over any remaining funds if you checked the rollover box when creating the fund.

1. To access the year end closeout of a fund, click Admin # Server Administration # Acquisitions # Funds.

2. Click Fund Propagation and Rollover.

3. Check the box adjacent to Perform Fiscal Year Close-Out Operation.

4. Notice that the context org unit reflects the context org unit that you selected at the top of the Funds screen.

5. If you want to perform the close-out operation on the context org unit and its child units, then check the box adjacent to Include Funds for Descendant Org Units.

6. Check the box adjacent to dry run if you want to test changes to the funds before they are enacted. Evergreen will generate a summary of the changes that would occur during the selected operations. No data will be changed.

7. Click Process.

8. Evergreen will begin the propagation process. Evergreen will make a clone of each fund, but it will increment the year by .

Providers

Providers are vendors. You can create a provider profile that includes contact information for the provider, holdings information, invoices, and other information.

Create a provider

1. To create a new provider, click Admin # Server Administration #Acquisitions # Providers.

2. Enter the provider name.

3. Create a code for the provider. No limits exist on the number of characters that can be entered in this field.

4. Select an owner from the drop down menu. The owner indicates the organizational units whose staff can use this provider. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).

The rule of parental inheritance applies to this list. See section .1 for more information.

5. Select a currency from the drop down menu. This drop down list is populated by the list of currencies available in the currency types.

6. A provider must be active in order for purchases to be made from that provider. To activate the provider, check the box adjacent to Active. To deactivate a vendor, uncheck the box.

7. Select a default claim policy from the drop down box. This list is derived from the claim policies that can be created

8. Select an EDI default. This list is derived from the EDI accounts that can be created.

9. Enter the provider's email address.

10. In the Fax Phone field, enter the provider's fax number.

11. In the holdings tag field, enter the tag in which the provider places holdings data.

12. In the phone field, enter the provider's phone number.

13. If prepayment is required to purchase from this provider, then check the box adjacent to prepayment required.

14. Enter the Standard Address Number (SAN) for your provider.

15. Enter the web address for the provider's website in the URL field.

16. Click Save.

Add contact and holdings information to providers

After you save the provider profile, the screen reloads so that you can save additional information about the provider. You can also access this screen by clicking the hyperlinked name of the provider on the Providers screen. The tabs allow you to add a provider address and contact, attribute definitions, and holding subfields. You can also view invoices associated with the provider.

1. Enter a Provider Address, and click Save.

Required fields for the provider address are: Street 1, city, state, country, post code. You may have multiple valid addresses.

2. Enter the Provider Contact, and click Save.

3. Your vendor may include information that is specific to your organization in MARC tags. You can specify the types of information that should be entered in each MARC tag. Enter attribute definitions to correlate MARC

tags with the information that they should contain in incoming vendor records. Some technical knowledge is required to enter XPath information.

4. You may have entered a holdings tag when you created the provider profile. You can also enter holdings subfields. Holdings subfields allow you to specify subfields within the holdings tag to which your vendor adds holdings information.

5. Click invoices to access invoices associated with a provider.

Edit a provider

Edit a provider just as you would edit a currency type.

You can delete providers only if no purchase orders have been assigned to them.

EDI

Many libraries use Electronic Data Interchange (EDI) accounts to order new acquisitions. In Evergreen 2.0, users can set up EDI accounts and manage EDI messages in the admin module. EDI messages and notes can be viewed in the acquisitions module.

The following fields are required to create an EDI account: host, username, password, path, and incoming directory.

EDI Accounts

Create EDI Accounts to communicate electronically with providers.

1. Create a label. The label allows you to differentiate between accounts for the same provider. No limits exist on the number of characters that can be entered in this field.

2. Enter a host. Your provider will provide you with the requisite FTP or SCP information.

3. Enter the username that has been supplied by your provider.

4. Enter the password that has been supplied by your provider.

5. Enter account information. This field enables you to add a supplemental password for entry to a remote system after log in has been completed. This field is optional for the ILS but may be required by your provider.

6. Select an owner from the drop down menu. The owner indicates the organizational units whose staff can use this EDI account. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).

The rule of parental inheritance applies to this list.

7. The Last Activity updates automatically with any inbound or outbound communication.

8. Select a provider from the drop down menu to whom this account belongs.

9. Enter a path. The path indicates the remote location on the server from which files are pulled in to the ILS.

10. Enter the incoming directory. This directory indicates the location on your local network to which the files download.

11. Enter the vendor account number supplied by your provider.

12. Enter the vendor account code supplied by your provider.

13. Click Save.

EDI Messages

The EDI messages screen displays all incoming and outgoing messages between the library and the vendor.

Claiming

Evergreen 2.0 provides minimal claiming functionality. Currently, all claiming is manual, but the admin module enables you to build claim policies and specify the action(s) that users should take to claim items.

Create a claim policy

The claim policy link enables you to name the claim policy and specify the organization that owns it.

1. To create a claim policy, click Admin # Server Administration # Acquisitions # Claim Policies.

2. Create a claim policy name. No limits exist on the number of characters that can be entered in this field.

3. Select an org unit from the drop down menu. The org unit indicates the organizational units whose staff can use this claim policy. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).

The rule of parental inheritance applies to this list.

4. Enter a description. No limits exist on the number of characters that can be entered in this field.

5. Click Save.

Create a claim type

The claim type link enables you to specify the reason for a type of claim.

1. To create a claim type, click Admin # Server Administration # Acquisitions # Claim types.

2. Create a claim type. No limits exist on the number of characters that can be entered in this field.

3. Select an org unit from the drop down menu. The org unit indicates the organizational units whose staff can use this claim type. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).

The rule of parental inheritance applies to this list.

4. Enter a description. No limits exist on the number of characters that can be entered in this field.

5. Click Save.

Create a claim event type

The claim event type describes the physical action that should occur when an item needs to be claimed. For example, the user should notify the vendor via email that the library is claiming an item.

1. To access the claim event types, click Admin # Server Administration # Acquisitions #Claim event type.

2. Enter a code for the claim event type. No limits exist on the number of characters that can be entered in this field.

3. Select an org unit from the drop down menu. The org unit indicates the organizational units whose staff can use this event type. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).

The rule of parental inheritance applies to this list.

4. Enter a description. No limits exist on the number of characters that can be entered in this field.

5. If this claim is initiated by the user, then check the box adjacent to Library Initiated.

Currently, all claims are initiated by a user. The ILS cannot automatically claim an issue.

6. Click Save.

Create a claim policy action

The claim policy action enables you to specify how long a user should wait before claiming the item.

1. To access claim policy actions, click Admin # Server Administration # Acquisitions #Claim Policy Actions.

2. Select an Action (Event Type) from the drop down menu.

3. Enter an action interval. This field indicates how long a user should wait before claiming the item.

4. In the Claim Policy ID field, select a claim policy from the drop down menu.

5. Click Save.

You can create claim cycles by adding multiple claim policy actions to a claim policy.

Invoice menus

Invoice menus allow you to create drop down menus that appear on invoices. You can create an invoice item type or invoice payment method.

Invoice item type

The invoice item type allows you to enter the types of additional charges that you can add to an invoice. Examples of additional charge types might include taxes or processing fees. Charges for bibliographic items are listed separately from these additional charges. A default list of charge types displays, but you can add custom charge types to this list. Invoice item types can also be used when adding non-bibliographic items to a purchase order. When invoiced, the invoice item type will copy from the purchase order to the invoice.

1. To create a new charge type, click Admin # Server Administration # Acquisitions # Invoice Item Type.

2. Click New Invoice Item Type.

3. Create a code for the charge type. No limits exist on the number of characters that can be entered in this field.

4. Create a label. No limits exist on the number of characters that can be entered in this field. The text in this field appears in the drop down menu on the invoice.

5. If items on the invoice were purchased with the monies in multiple funds, then you can divide the additional charge across funds. Check the box adjacent to Prorate? if you want to prorate the charge across funds.

6. Click Save.

Invoice payment method

The invoice payment method allows you to predefine the type(s) of invoices and payment method(s) that you accept. The text that you enter in the admin module will appear as a drop down menu in the invoice type and payment method fields on the invoice.

1. To create a new invoice payment method, click Admin # Server Administration # Acquisitions # Invoice Payment Method.

2. Click New Invoice Payment Method.

3. Create a code for the invoice payment method. No limits exist on the number of characters that can be entered in this field.

4. Create a name for the invoice payment method. No limits exist on the number of characters that can be entered in this field. The text in this field appears in the drop down menu on the invoice.

5. Click Save.

Distribution Formulas

Distribution formulas allow you to specify the number of copies that should be distributed to specific branches. You can create and reuse formulas as needed.

Create a distribution formula

1. Click Admin # Server Administration # Acquisitions #Distribution Formulas.
2. Click New Formula.
3. Enter a Formula Name. No limits exist on the number of characters that can be entered in this field.
4. Choose a Formula Owner from the drop down menu. The Formula Owner indicates the organizational units whose staff can use this formula. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).

The rule of parental inheritance applies to this list.

5. Ignore the Skip Count field which is currently not used.
6. Click Save.
7. Click New Entry.
8. Select an Owning Library from the drop down menu. This indicates the branch that will receive the items. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).
9. Select a Shelving Location from the drop down menu.
10. In the Item Count field, enter the number of items that should be distributed to the branch. You can enter the number or use the arrows on the right side of the field.
11. Click Apply Changes. The screen will reload.
12. To view the changes to your formula, click Admin # Server Administration # Acquisitions # Distribution Formulas. The item_count will reflect the entries to your distribution formula.

To edit the Formula Name, click the hyperlinked name of the formula in the top left corner. A pop up box will enable you to enter a new formula name.

Edit a distribution formula

To edit a distribution formula, click the hyperlinked title of the formula.

Line item features

Line item alerts are predefined text that can be added to line items that are on selection lists or purchase orders. You can define the alerts from which staff can choose. Line item alerts appear in a pop up box when the line item, or any of its copies, are marked as received.

Create a line item alert

1. To create a line item alert, click Administration # Server Administration # Acquisitions # Line Item Alerts.
2. Click New Line Item Alert Text.
3. Create a code for the text. No limits exist on the number of characters that can be entered in this field.
4. Create a description for the text. No limits exist on the number of characters that can be entered in this field.
5. Select an owning library from the drop down menu. The owning library indicates the organizational units whose staff can use this alert. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units).
6. Click Save.

Line Item MARC Attribute Definitions

Line item attributes define the fields that Evergreen needs to extract from the bibliographic records that are in the acquisitions database to display in the catalog. Also, these attributes will appear as fields in the New Brief Record interface. You will be able to enter information for the brief record in the fields where attributes have been defined.

Cancel/Suspend reasons

The Cancel reasons link enables you to predefine the reasons for which a line item or a PO can be cancelled. A default list of reasons appears, but you can add custom reasons to this list. Applying the cancel reason will prevent the item from appearing in a claims list and will allow you to cancel debits associated with the purchase. Cancel reasons also enable you to suspend or delay a purchase. For example, you could create a cancel reason of “back ordered,” and you could choose to keep the debits associated with the purchase.

Create a cancel/suspend reason

1. To add a new cancel reason, click Administration # Server Administration # Acquisitions # Cancel reasons.
2. Click New Cancel Reason.
3. Select a using library from the drop down menu. The using library indicates the organizational units whose staff can use this cancel reason. This menu is populated with the shortnames that you created for your libraries in the organizational units tree (See Admin # Server Administration # Organizational Units.)
4. Create a label for the cancel reason. This label will appear when you select a cancel reason on an item or a PO.
5. Create a description of the cancel reason. This is a free text field and can be comprised of any text of your choosing.

6. If you want to retain the debits associated with the cancelled purchase, click the box adjacent to Keep Debits?

7. Click Save.

Acquisitions Permissions in the Admin module

Several setting in the Library Settings area of the Admin module pertain to functions in the Acquisitions module. You can access these settings by clicking Admin # Local Administration #Library Settings Editor.

- CAT: Delete bib if all copies are deleted via Acquisitions lineitem cancellation – If you cancel a line item, then all of the on order copies in the catalog are deleted. If, when you cancel a line item, you also want to delete the bib record, then set this setting to TRUE.
- Default circulation modifier – This modifier would be applied to items that are created in the acquisitions module
- Default copy location – This copy location would be applied to items that are created in the acquisitions module
- Fund Spending Limit for Block - When the amount remaining in the fund, including spent money and encumbrances, goes below this percentage, attempts to spend from the fund will be blocked.
- Fund Spending Limit for Warning - When the amount remaining in the fund, including spent money and encumbrances, goes below this percentage, attempts to spend from the fund will result in a warning to the staff.
- Temporary barcode prefix - Temporary barcode prefix for items that are created in the acquisitions module
- Temporary call number prefix - Temporary call number prefix for items that are created in the acquisitions module

Chapter 15. Booking Module Administration

Adapted with permission from original material by the Evergreen Community

Abstract

The Evergreen booking module is included in Evergreen 1.6.1.x and above.The following documentation will include information about making cataloged items bookable; making non-bibliographic items bookable; and setting permissions in the booking module for staff.

Make a Cataloged Item Bookable in Advance

If their permission settings allow, staff members can make items bookable. Staff members can do this in advance of a booking request, or they can do it on the fly.

If you know in advance of the request that an item will need to be booked, you can make the item bookable.

1. In the staff client, select Search → Search the Catalog
2. Begin a title search to find an item.
3. Click the title of the item that you want to book.
4. The Record Summary will appear. In this view you can see information about the item and its locations. Click Actions for this Record → Holdings Maintenance in the top right corner of the screen.
5. The Holdings Maintenance screen will appear. In this screen, you can view the volumes and copies of an item avaialable at each branch. To view the barcodes and other information for each copy, click the arrow adjacent to the branch with the copy that you need to view. Click on successive arrows until you find the copy that you need to view.
6. Select the item that you want to make bookable. Right click to open the menu, and click Make Item Bookable.
7. The item has now been added to the list of resources that are bookable. To book the item, return to the Record Summary, and proceed with booking..

In Evergreen 1.6.1, there is no way to make an item “unbookable” after it has been made bookable and has been reserved. The Delete Selected button on this screen deletes the resource from the screen, but the item will be able to be booked after it has been returned.

Make a Cataloged Item Bookable On the Fly

If a patron wants to book an item immediately that does not have bookable status, you can book the item on the fly if you have the appropriate permissions.

1. Follow steps one through five in the section called “Make a Cataloged Item Bookable in Advance”.

2. Select the item that you want to make bookable. Right click to open the menu, and click Book Item Now.

3. A Reservations screen will appear in a new tab, and you can make the reservation.

Create a Bookable Status for Non-Bibliographic Items

Staff with the required permissions can create a bookable status for non-bibliographic items. For example, staff can book conference rooms or laptops. You will be able to create types of resources, specify the names of individual resources within each type, and set attributes to describe those resources. You can then bring the values together through the Resource Attribute Map.

1. First, create the type of resource that you want to make bookable. Select Admin → Server Administration → Booking → Resource Types.

2. A list of resource types will appear. You may also see titles of cataloged items on this screen if they were added using the Make Item Bookable or Book Now links. You should not attempt to add cataloged items on this screen; it is best to use the aforementioned links to make those items bookable. In this screen, you will create a type of resource.

3. In the right corner, click New Resource Type.

4. A box will appear in which you will create a type of resource. In this box, you can set fines, determine “elbow room” periods between reservations on this type of resource, and indicate if this type of resource can be transferred to another library. Click Save when you have entered the needed information.

5. After you click Save, the box will disappear. Refresh the screen to see the item that you have added.

6. Next, set the attributes for the type of resource that you have created. Select Server Administration → Booking → Resource Attributes.

7. Click New Resource Attribute.

8. A box will appear in which you can add the attributes of the resource. Attributes are descriptive information that is provided to the staff member when the booking request is made. For example, an attribute of the projector may be a cart that allows for its transportation. Other attributes might be number of seats available in a room, or MAC or PC attributes for a laptop. Click Save when the necessary information has been entered.

9. The box will disappear. Refresh the screen to see the added attribute.

10. Next, add the values for the resource attributes. A value can be a number, yes/no, or any other meaningful information. Select Server Administration → Booking → Resource Attribute Values.

11. Select New Resource Attribute Value.

12. A pop up box will appear. Select the Resource Attribute from the drop down box. Add the value. You can add multiple values for this field. Click Save when the required information has been added.

13. If you refresh the screen, the attribute value may not appear, but it has been saved.

14. Next, identify the specific objects that are associated with this resource type. Click Admin → Server Administration → Booking → Resources.

15. Click New Resource.

16. A pop-up box will appear. Add information for the resource and click Save. Repeat this process for each resource.

17. Refresh the screen, and the resource(s) that you added will appear.

18. Finally, use Resource Attribute Maps to bring together the resource and its attributes. Select Admin → Server Administration → Booking → Resource Attribute Maps.

19. Select New Resource Attribute Map

20. Select the resource that you want to match with its attributes, then click Save. Repeat for all applicable resources.

21. You have now created bookable, non-bibliographic resource(s) with attributes.

Setting Booking Permissions

Administrators can set permissions so that staff members can view reservations, make reservations, and make bibliographic or non-bibliographic items bookable.

If a staff member attempts to book an item for which they do not have the appropriate permissions, they will receive an error message.

To set permissions, select Admin → Server Administration → Permissions.

Staff members should be assigned the following permissions to do common tasks in the booking module. These permissions could be assigned to front line staff members, such as circulation staff. Permissions with an asterisk (*****) are already included in the **Staff** permission group. All other booking permissions must be applied individually.

- **View Reservations:** VIEW_TRANSACTION*
- **Use the pull list:** RETRIEVE_RESERVATION_PULL_LIST
- **Capture reservations:** CAPTURE_RESERVATION
- **Assist patrons with pickup and return:** VIEW_USER*
- **Create/update/delete reservations:** ADMIN_BOOKING_RESERVATION

The following permissions allow users to do more advanced tasks, such as making items bookable, booking items on the fly, and creating non-bibliographic resources for booking.

- **Create/update/delete booking resource type:** ADMIN_BOOKING_RESOURCE_TYPE
- **Create/update/delete booking resource attributes:** ADMIN_BOOKING_RESOURCE_ATTR

- **Create/update/delete booking resource attribute values:** ADMIN_BOOKING_RESOURCE_ATTR_VALUE
- **Create/update/delete booking resource:** ADMIN_BOOKING_RESOURCE
- **Create/update/delete booking resource attribute maps:** ADMIN_BOOKING_RESOURCE_ATTR_MAP

In addition to having the permissions listed above, staff members will need a valid working location in their profiles. This should be done when registering new staff members.

Part V. Reports

Reports are a powerful tool in Evergreen and can be used for statistical comparisons or collection maintenance. The following part covers everything dealing with reports from starting the reporter daemon to viewing reports your library has created. The range of topics in this part is quite broad and different chapters will be useful to different roles in an Evergreen library system.

Chapter 16. Starting and Stopping the Reporter Daemon

Before you can view reports, the Evergreen administrator must start the reporter daemon from the command line of the Evergreen server.

The reporter daemon periodically checks for requests for new reports or scheduled reports and gets them running.

Starting the Reporter Daemon

To start the reporter daemon, run the following command as the opensrf user:

```
clark-kent.pl --daemon
```

You can also specify other options:

- sleep=interval : number of seconds to sleep between checks for new reports to run; defaults to 10
- lockfile=filename : where to place the lockfile for the process; defaults to /tmp/reporter-LOCK
- concurrency=integer : number of reporter daemon processes to run; defaults to 1
- boostrap=filename : OpenSRF bootstrap configuration file; defaults to /openils/conf/opensrf_core.xml

The open-ils.reporter process must be running and enabled on the gateway before the reporter daemon can be started.

Remember that if the server is restarted, the reporter daemon will need to be restarted before you can view reports unless you have configured your server to start the daemon automatically at start up time.

Stopping the Reporter Daemon

To stop the reporter daemon, you have to kill the process and remove the lockfile. Assuming you're running just a single process and that the lockfile is in the default location, perform the following commands as the opensrf user:

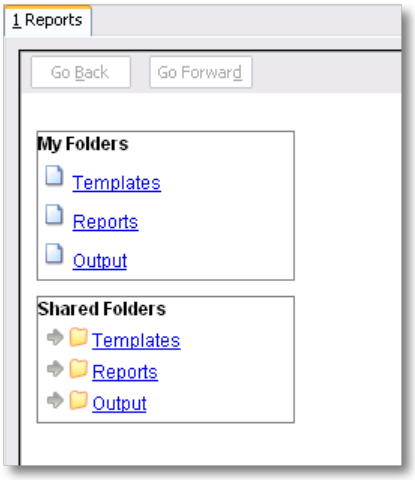
```
kill `ps wax | grep "Clark Kent" | grep -v grep | cut -b1-6`
```

```
rm /tmp/reporter-LOCK
```

Chapter 17. Folders

There are three main components to reports: Templates, Reports, and Output. Each of these components must be stored in a folder. Folders can be private (accessible to your login only) or shared with other staff at your library, other libraries in your system or consortium. It is also possible to selectively share only certain folders and/or subfolders.

There are two parts to the folders pane. The My Folders section contains folders created with your Evergreen account. Folders that other users have shared with you appear in the Shared Folders section under the username of the sharing account.



Creating Folders

Whether you are creating a report from scratch or working from a shared template you must first create at least one folder.

The steps for creating folders are similar for each reporting function. It is easier to create folders for templates, reports, and output all at once at the beginning, though it is possible to do it before each step. This example demonstrates creating a folder for a template.

1. Click on Templates in the My Folders section.
2. Name the folder. Select Share or Do not share from the dropdown menu.
3. If you want to share your folder, select who you want to share this folder with from the dropdown menu.
4. Click Create Sub Folder.
5. Click OK.
6. Next, create a folder for the report definition to be saved to. Click on Reports.
7. Repeat steps 2-5 to create a Reports folder also called Circulation.
8. Finally, you need to create a folder for the report's output to be saved in. Click on Output.
9. Repeat steps 2-5 to create an Output folder named Circulation.

Using a parallel naming scheme for folders in Templates, Reports, and Output helps keep your reports organized and easier to find

The folders you just created will now be visible by clicking the arrows in My Folders. Bracketed after the folder name is whom the folder is shared with. For example, Circulation (BNCLF) is shared with the North Coast Library Federation. If it is not a shared folder there will be nothing after the folder name. You may create as many folders and sub-folders as you like.

Managing Folders

Once a folder has been created you can change the name, delete it, create a new subfolder, or change the sharing settings. This example demonstrates changing a folder name; the other choices follow similar steps

1. Click on the folder that you wish to rename.
2. Click Manage Folder.
3. Select Change folder name from the dropdown menu and click Go.
4. Enter the new name and click Submit
5. Click OK.
6. You will get a confirmation box that the Action Succeeded. Click OK.

Chapter 18. Creating Templates

Once you have created a folder, the next step in building a report is to create or clone a template. Templates allow you to run a report more than once without building it anew every time, by changing definitions to suit current requirements. For example, you can create a shared template that reports on circulation at a given library. Then, other libraries can use your template and simply select their own library when they run the report.

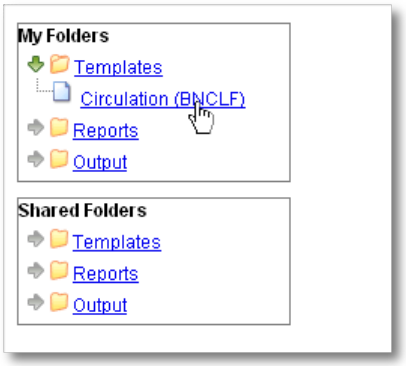
It may take several tries to refine a report to give the output that you want. It can be useful to plan out your report on paper before getting started with the reporting tool. Group together related fields and try to identify the key fields that will help you select the correct *source*.

It may be useful to create complex queries in several steps. For example, first add all fields from the table at the highest source level. Run a report and check to see that you get results that seem reasonable. Then clone the report, add any filters on fields at that level and run another report. Then drill down to the next table and add any required fields. Run another report. Add any filters at that level. Run another report. Continue until you've drilled down to all the fields you need and added all the filters. This might seem time consuming and you will end up cloning your initial report several times. However, it will help you to check the correctness of your results, and will help to debug if you run into problems because you will know exactly what changes caused the problem. Also consider adding extra fields in the intermediate steps to help you check your results for correctness.

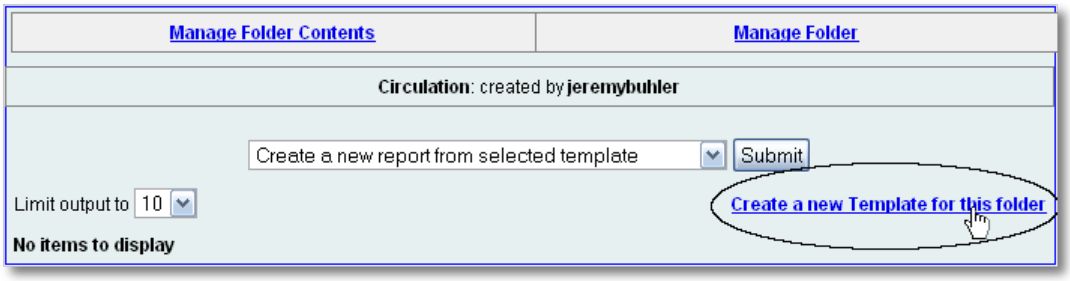
This example illustrates creating a template for circulation statistics. This is an example of the most basic template that you can create. The steps required to create a template are the same every time, but the tables chosen, how the data is transformed and displayed, and the filters used will vary depending on your needs.

Choosing Report Fields

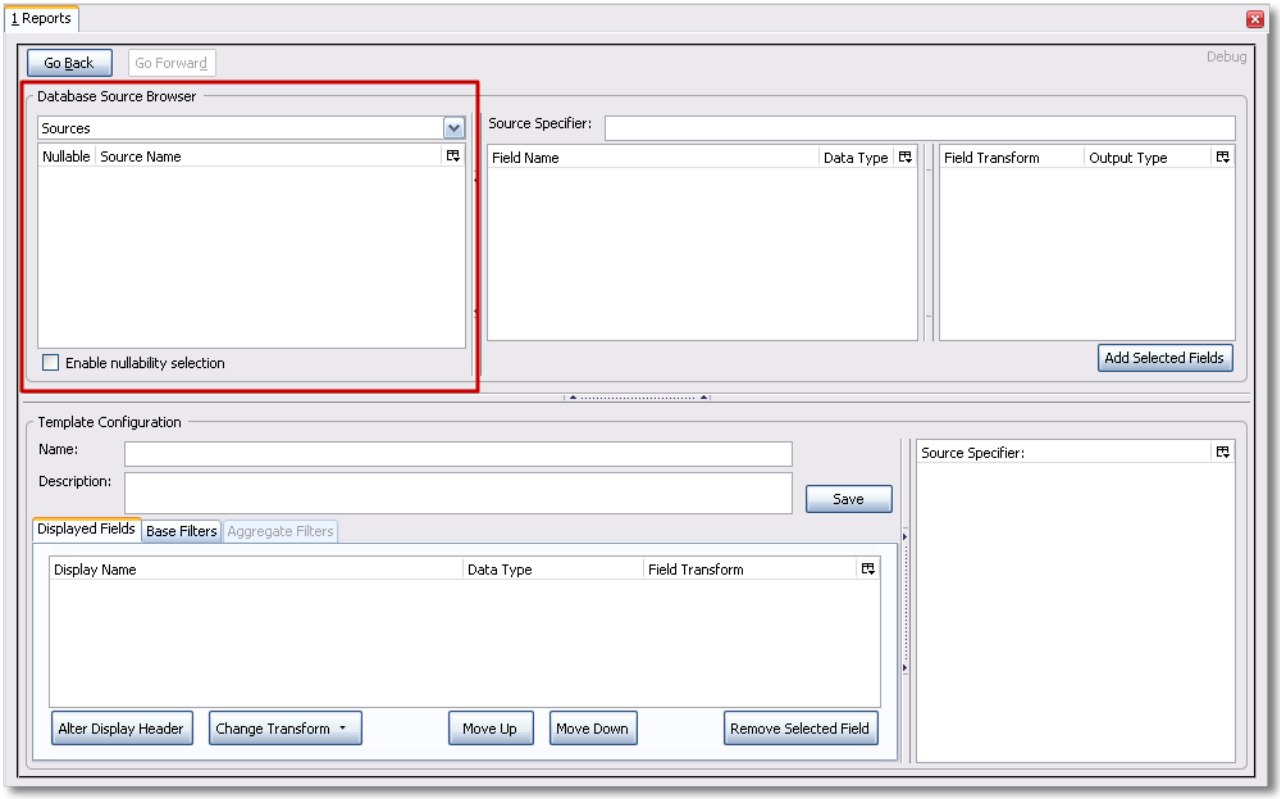
1. Click on the My Folder template folder where you want the template to be saved.



2. Click on Create a new Template for this folder.

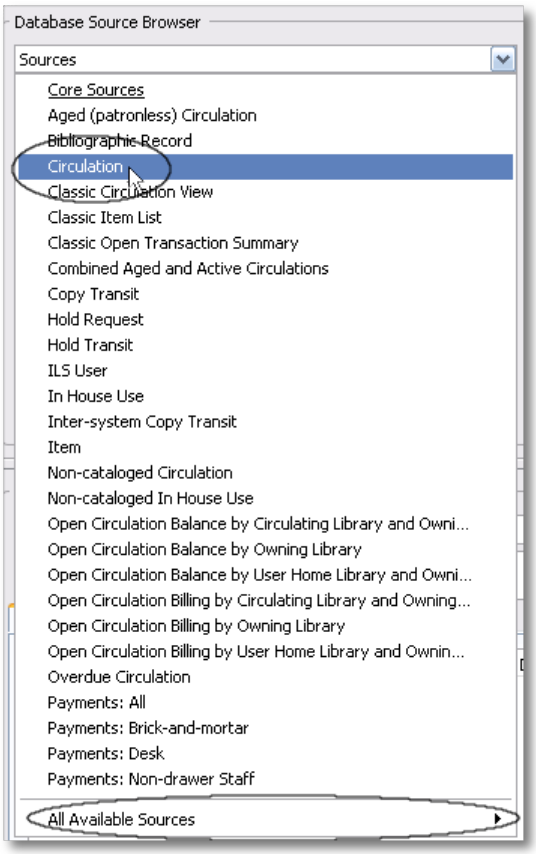


3. You can now see the template creating interface. The upper half of the screen is the Database Source Browser. The top left hand pane contains the database Sources drop-down list. This is the list of tables available as a starting point for your report. Commonly used sources are Circulation (for circ stats and overdue reports), ILS User (for patron reports), and Item (for reports on a library's holdings).

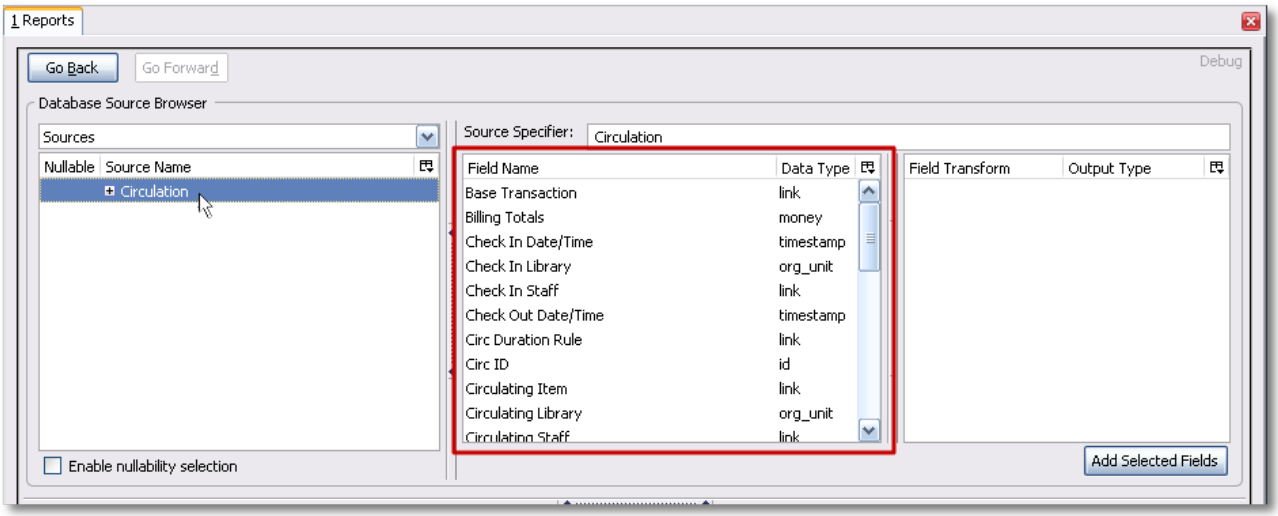


The Enable source nullability checkbox below the sources list is for advanced reporting and should be left unchecked by default.

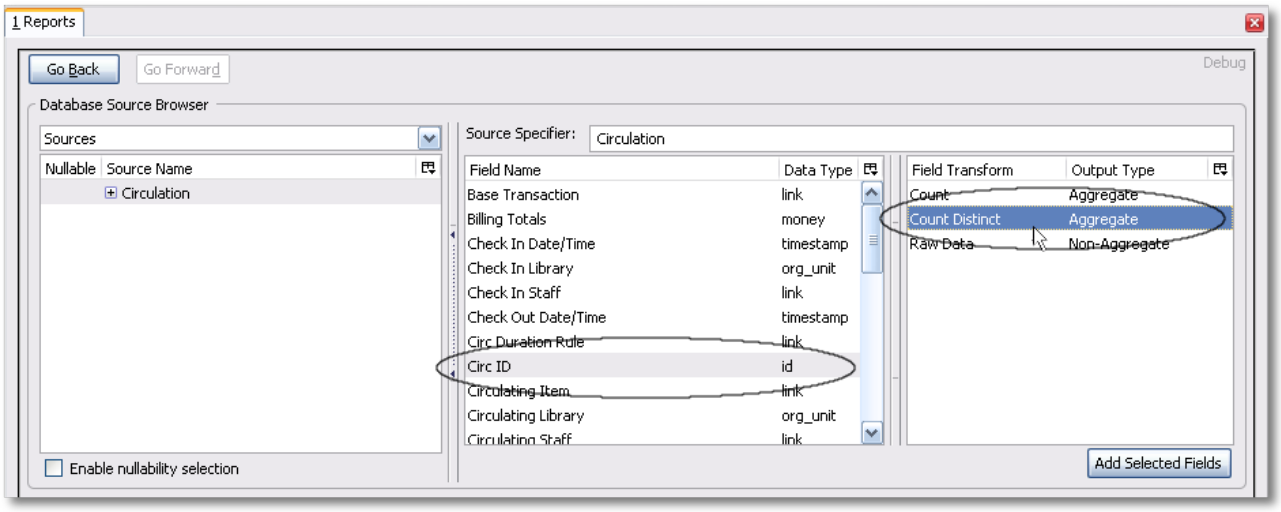
4. Select Circulation in the Sources dropdown menu. Note that the Core Sources for reporting are listed first, however it is possible to access all available sources at the bottom of this dropdown menu. You may only specify one source per template.



5. Click on Circulation to retrieve all the field names in the Field Name pane. Note that the Source Specifier (above the middle and right panes) shows the path that you took to get to the specific field.

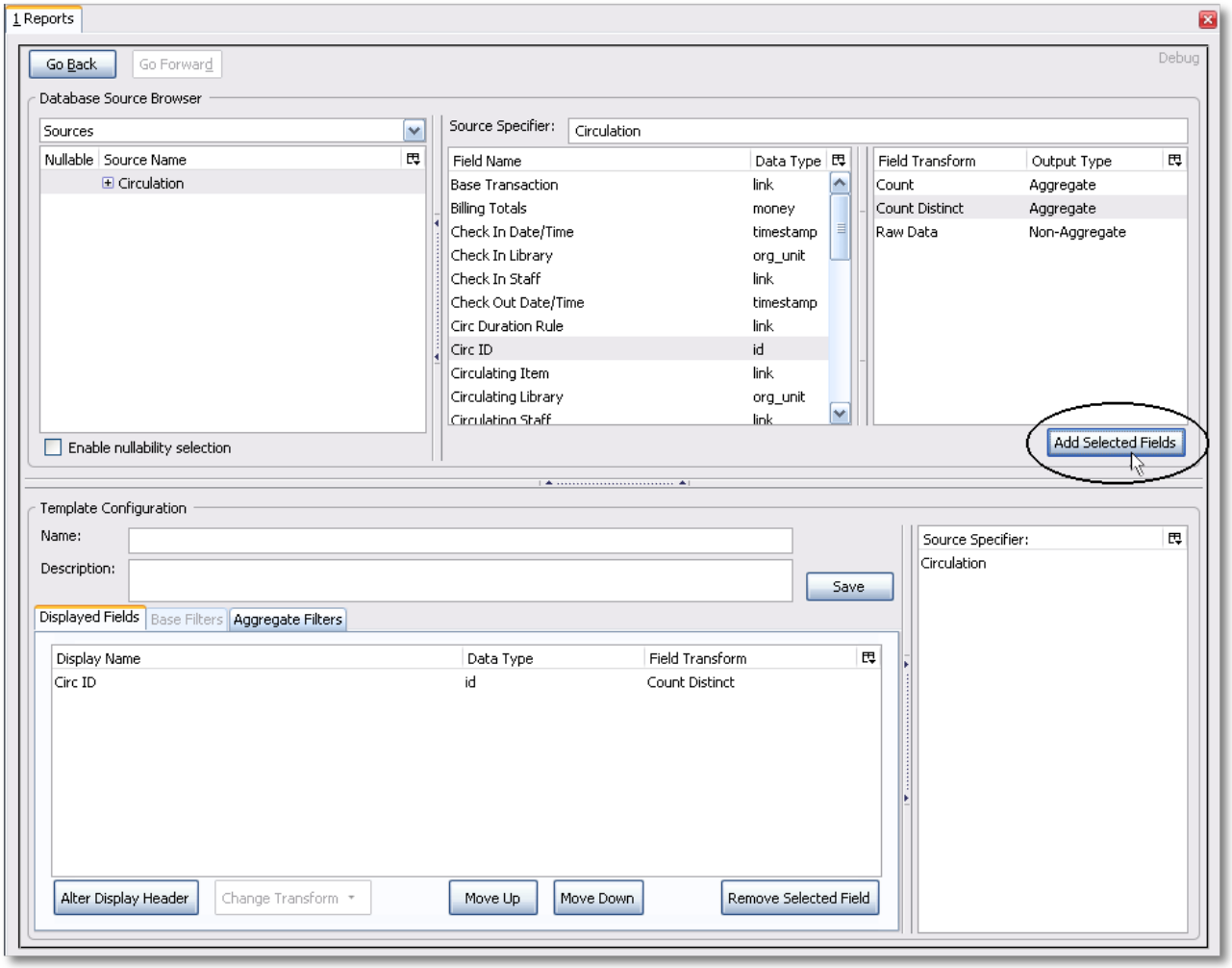


6. Select Circ ID in the middle Field Name pane, and Count Distinct from the right Field Transform pane. The Field Transform pane is where you choose how to manipulate the data from the selected fields. You are counting the number of circulations.

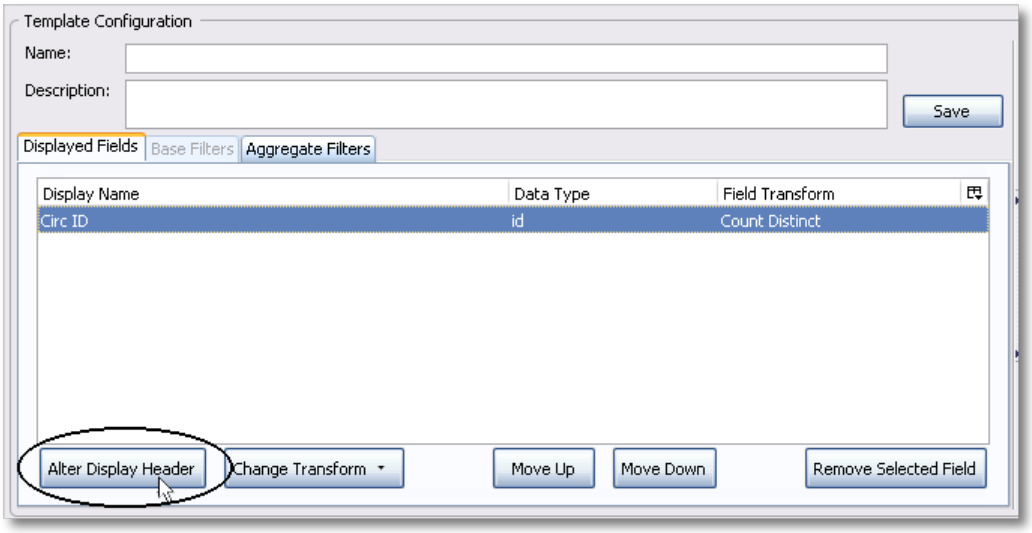


Field Transforms have either an Aggregate or Non-Aggregate output type. See the section called “Field Transforms” for more about Count, Count Distinct, and other transform options.

7. Click Add Selected Fields underneath the Field Transform pane to add this field to your report output. Note that Circ ID now shows up in the bottom left hand pane under the Displayed Fields tab.

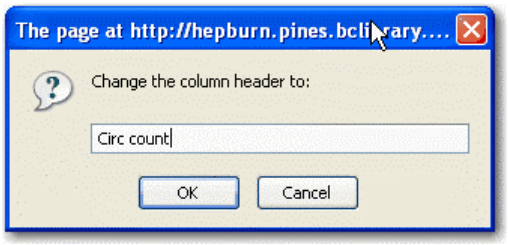


8. Circ ID will be the column header in the report output. You can rename default display names to something more meaningful. To do so in this example, select the Circ ID row and click Alter Display Header.



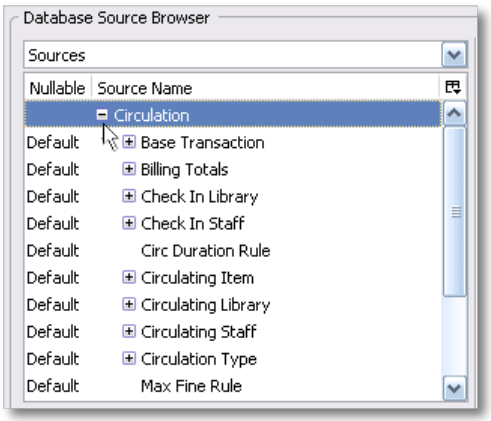
Double-clicking on the displayed field name is a shortcut to altering the display header.

9. Type in the new column header name, for example *Circ count* and click OK.

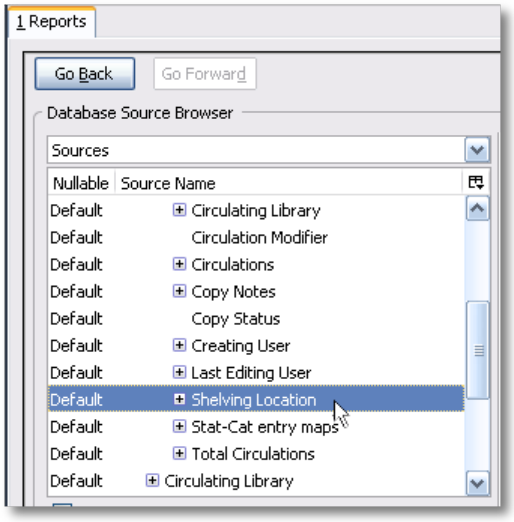


10. Add other data to your report by going back to the Sources pane and selecting the desired fields. In this example, we are going to add Circulating Item → Shelving Location to further refine the circulation report.

    In the top left hand Sources pane, expand Circulation. Depending on your computer you will either click on the + sign or on an arrow to expand the tree.

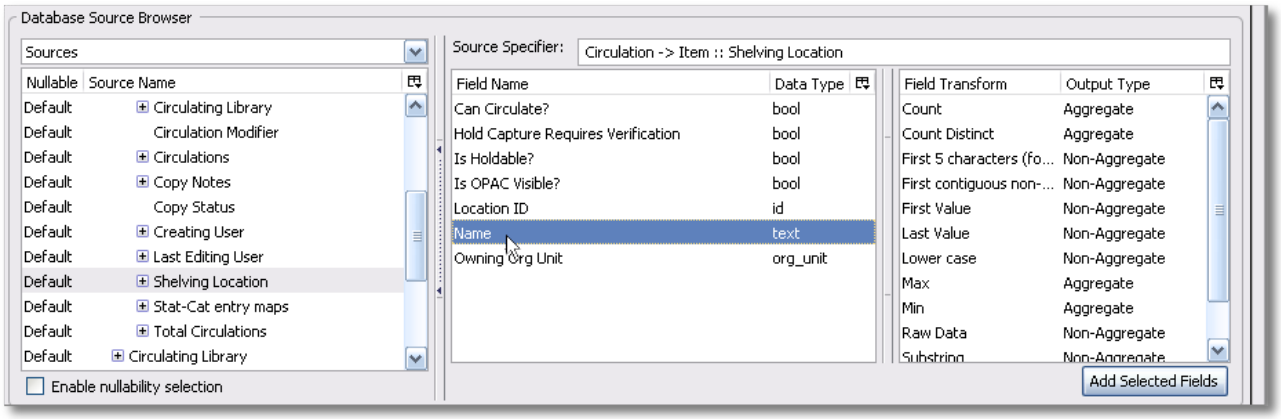


11. Click on the + or arrow to expand Circulating Item. Select Shelving Location.

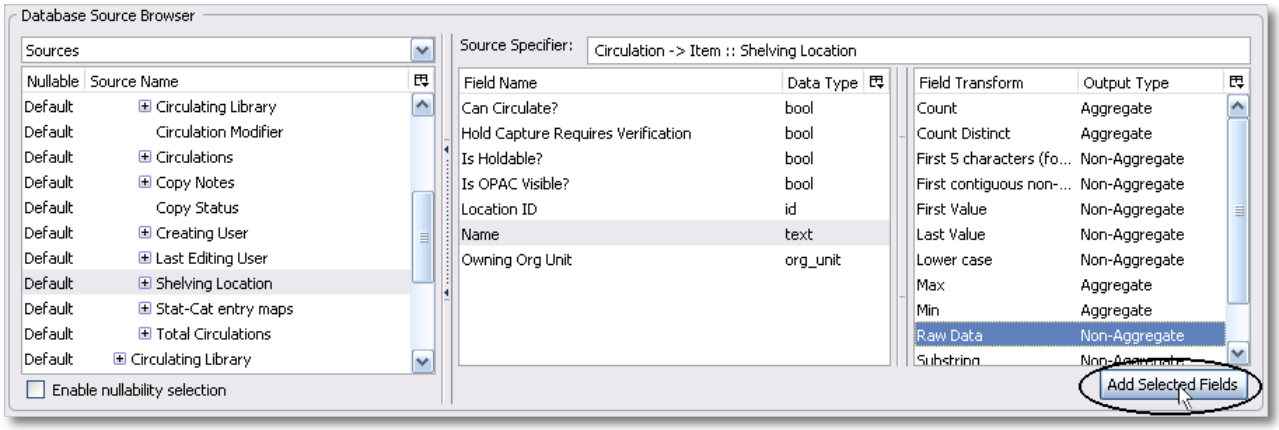


When you are creating a template take the shortest path to the field you need in the left hand Sources pane. Sometimes it is possible to find the same field name further in the file structure, but the shortest path is the most efficient.

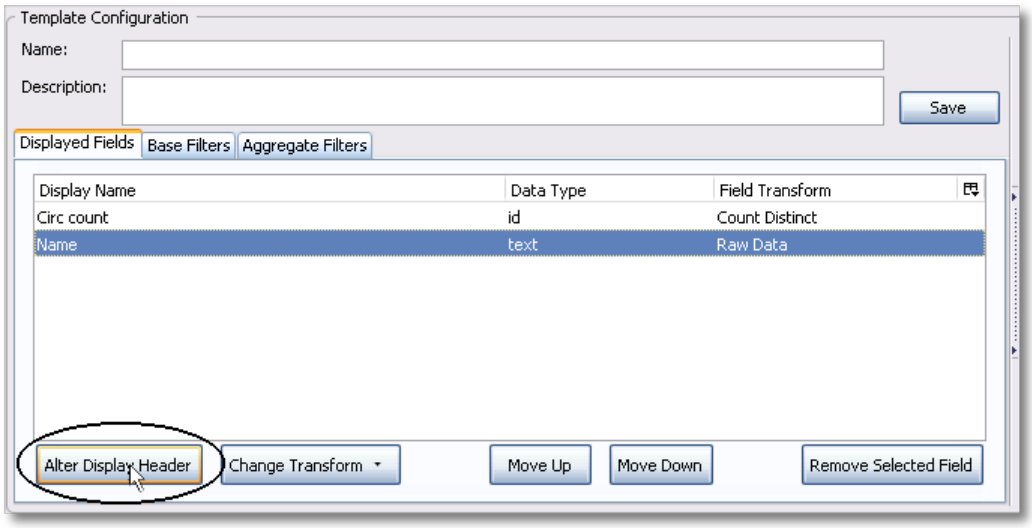
12. In the Field Name pane select Name.



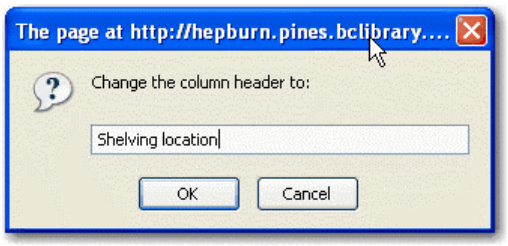
13. In the upper right Field Transform pane, select Raw Data and click Add Selected Fields. Use Raw Data when you do not wish to transform field data in any manner.



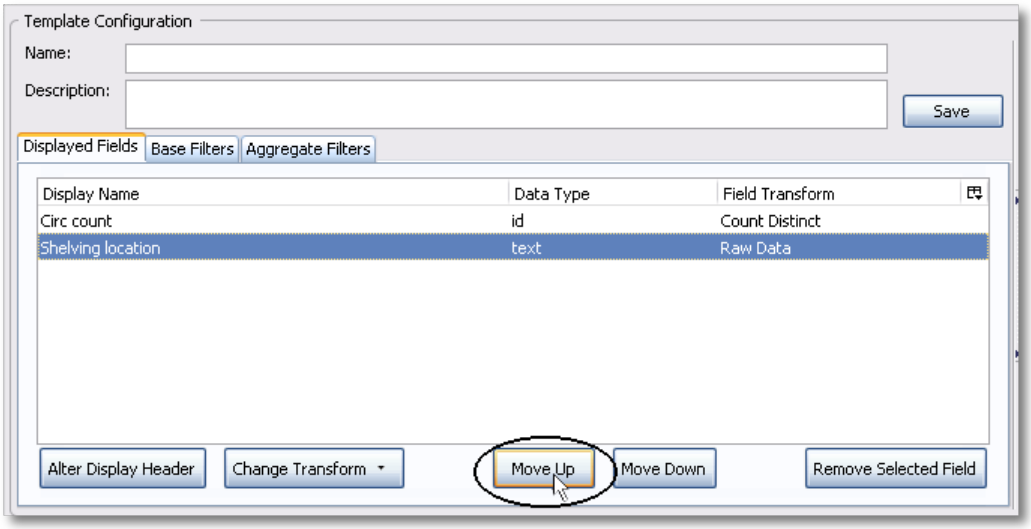
14. Name will appear in the bottom left pane. Select the Name row and click Alter Display Header.



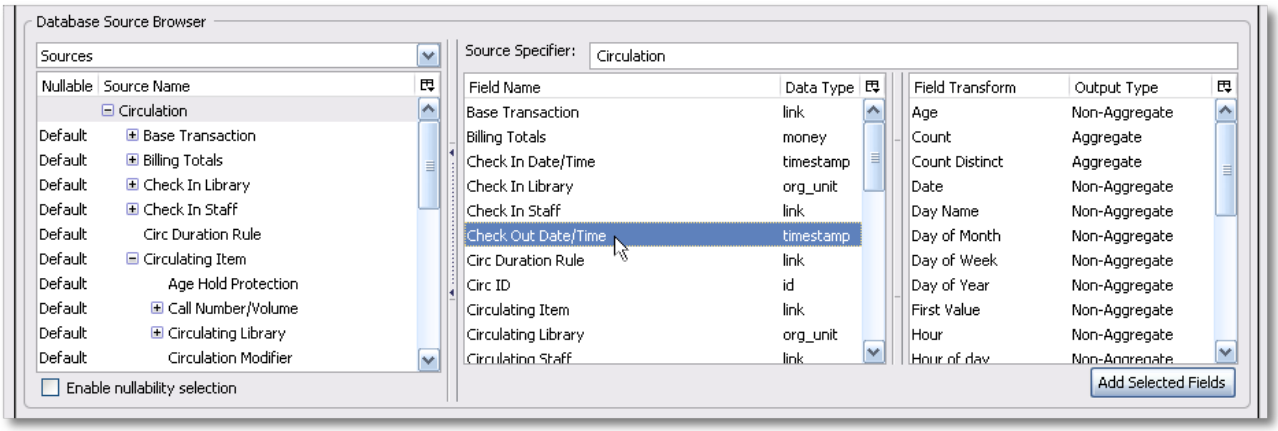
15. Enter a new, more descriptive column header, for example, *Shelving location*. Click OK.



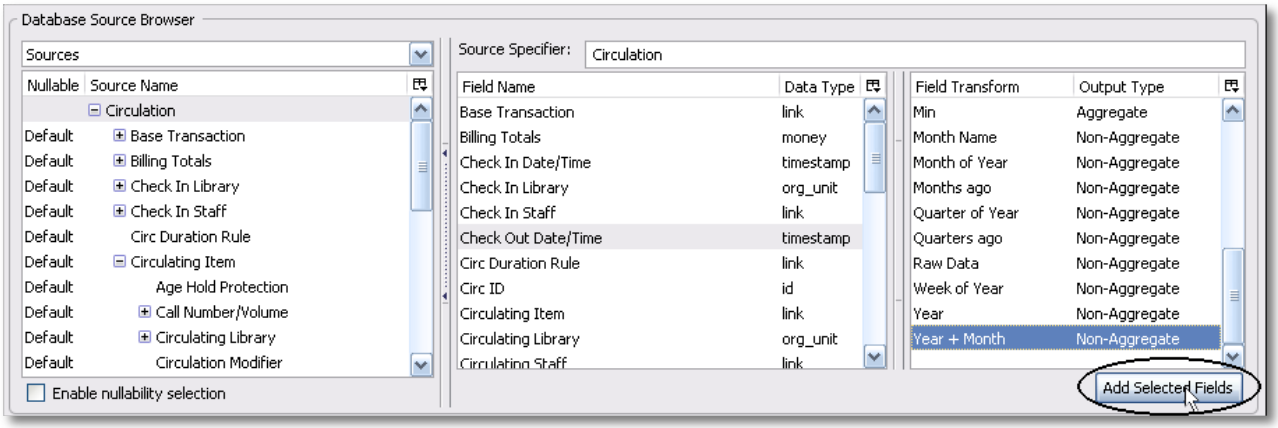
16. Note that the order of rows (top to bottom) will correspond to the order of columns (left to right) on the final report. Select Shelving location and click on Move Up to move Shelving location before Circ count.



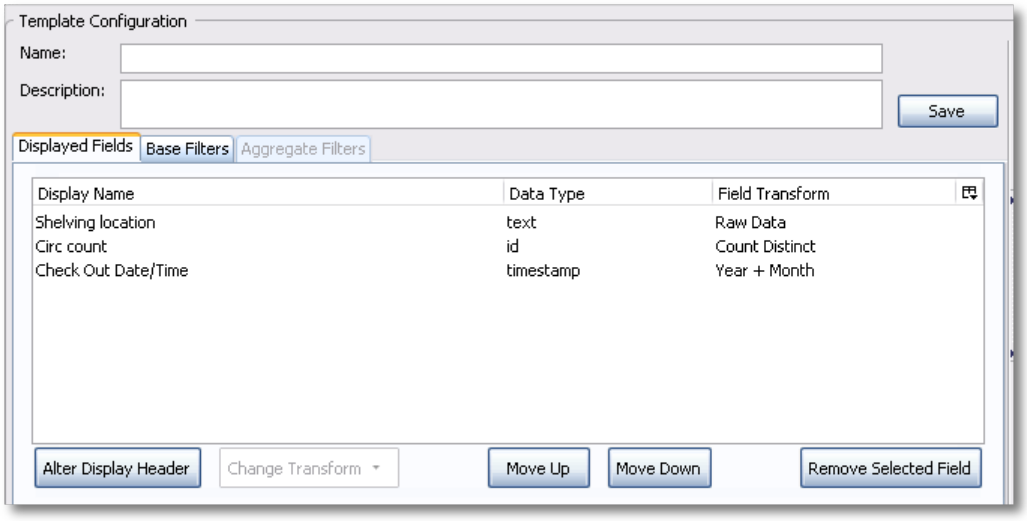
17. Return to the Sources pane to add more fields to your template. Under Sources click Circulation, then select Check Out Date/Time from the middle Field Name pane.



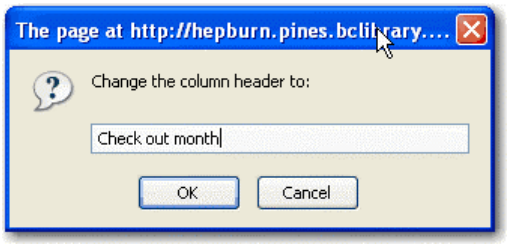
18. Select Year + Month in the right hand Field Transform pane and click Add Selected Fields



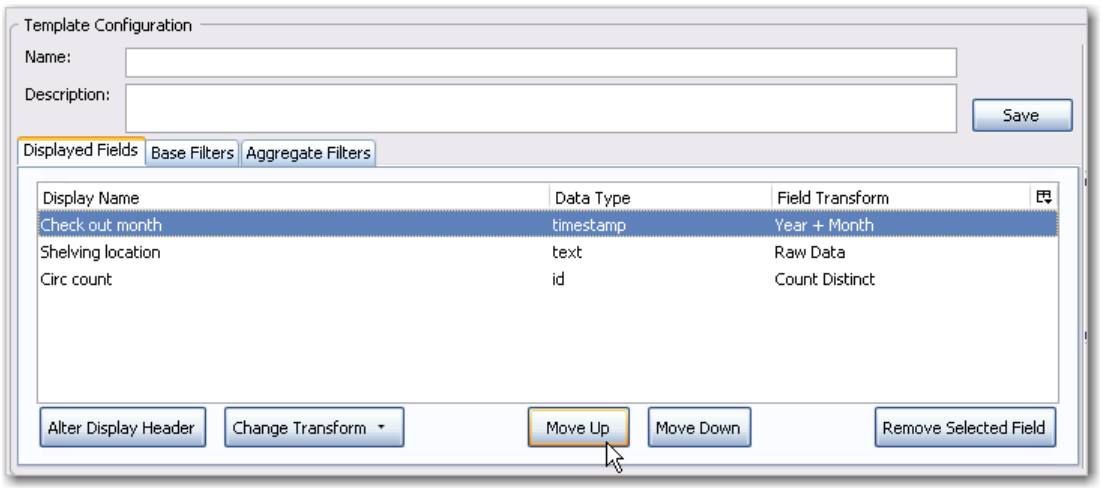
19. Check Out Date/Time will appear in the Displayed Fields pane. In the report it will appear as a year and month (YYYY-MM) corresponding to the selected tranform.



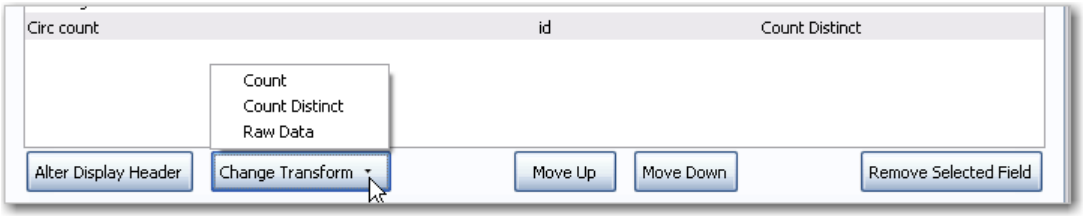
20. Select the Check Out Date/Time row. Click Alter Display Header and change the column header to *Check out month*.



21. Move Check out month to the top of the list using the Move Up button, so that it will be the first column in an MS Excel spreadsheet or in a chart. Report output will sort by the first column.



Note the Change Transform button in the bottom left hand pane. It has the same function as the upper right Field Transform pane for fields that have already been added.



Applying Filters

Evergreen reports access the entire database, so to limit report output to a single library or library system you need to apply filters.

After following the steps in the previous section you will see three fields in the bottom left hand Template Configuration pane. There are three tabs in this pane: Displayed Fields (covered in the previous section), Base Filters and Aggregate Filters. A filter allows you to return only the results that meet the criteria you set.

Base Filters apply to non-aggregate output types, while Aggregate Filters are used for aggregate types. In most reports you will be using the Base Filters tab. For more information on aggregate and non-aggregate types see the section called “Field Transforms”.

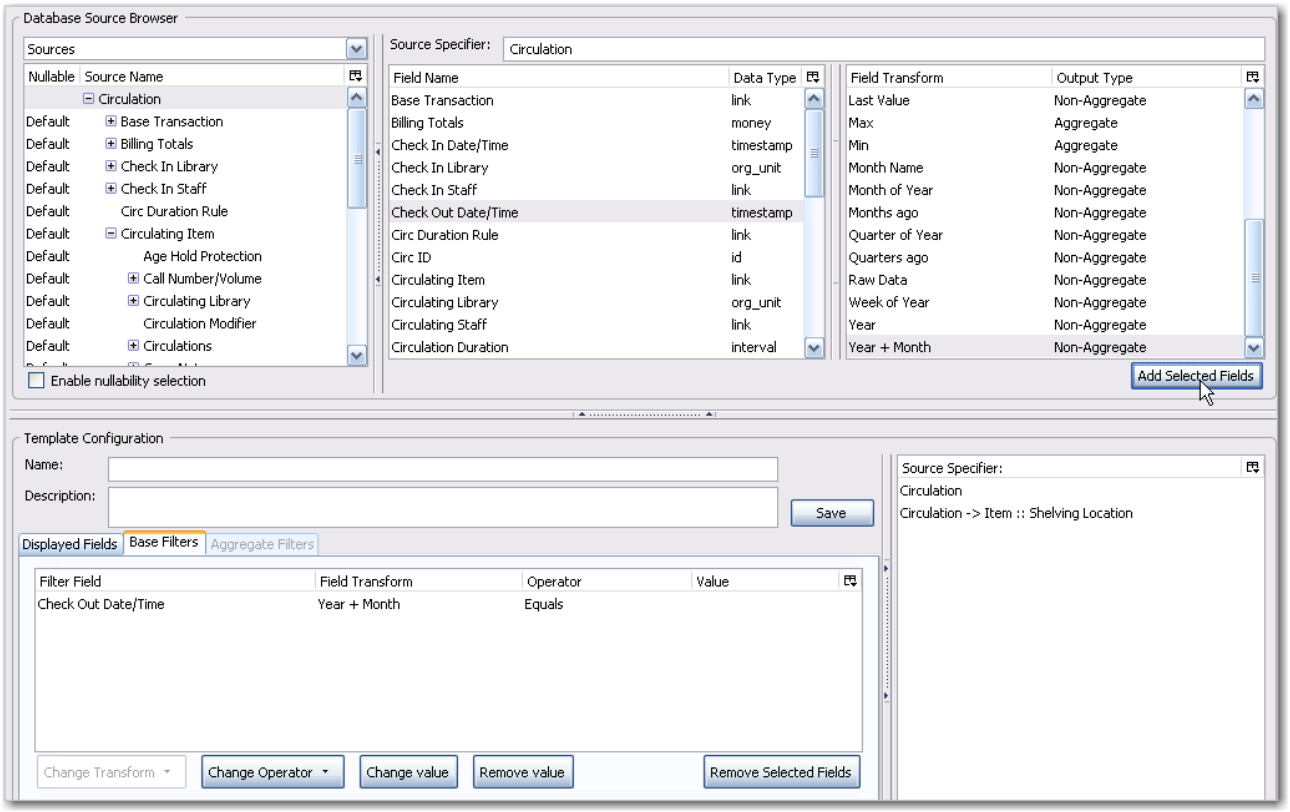
There are many available operators when using filters. Some examples are *Equals*, *In list*, *is NULL*, *Betwee*, *Greater than or equal to*, and so on. *In list* is the most flexible operator, and in this case will allow you flexibility when running a report from this template. For example, it would be possible to run a report on a list of timestamps (in this case will be trimmed to year and month only), run a report on a single month, or run a report comparing two months. It is also possible to set up recurring reports to run at the end of each month.

In this example we are going to use a *Base Filter* to filter out one library’s circulations for a specified time frame. The time frame in the template will be configured so that you can change it each time you run the report.

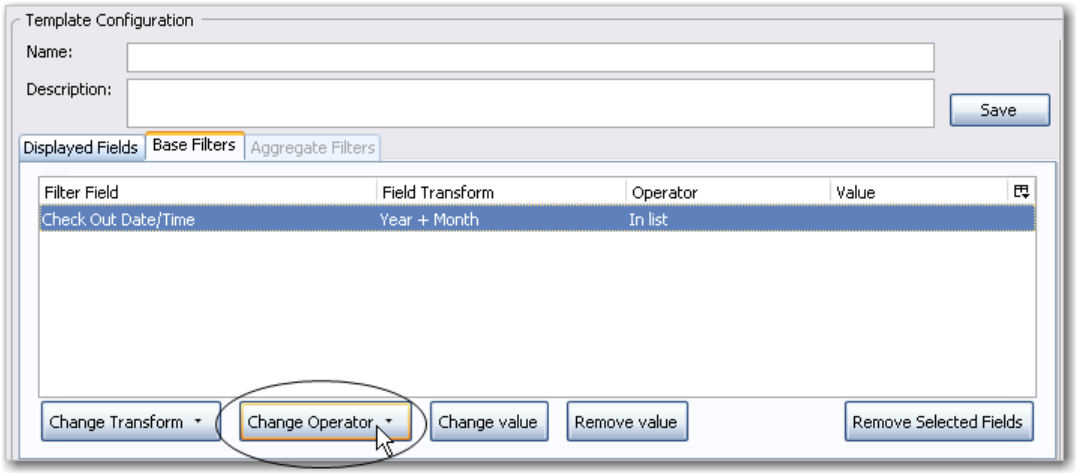
Using Base Filters

1. Select the Base Filters tab in the bottom Template Configuration pane.

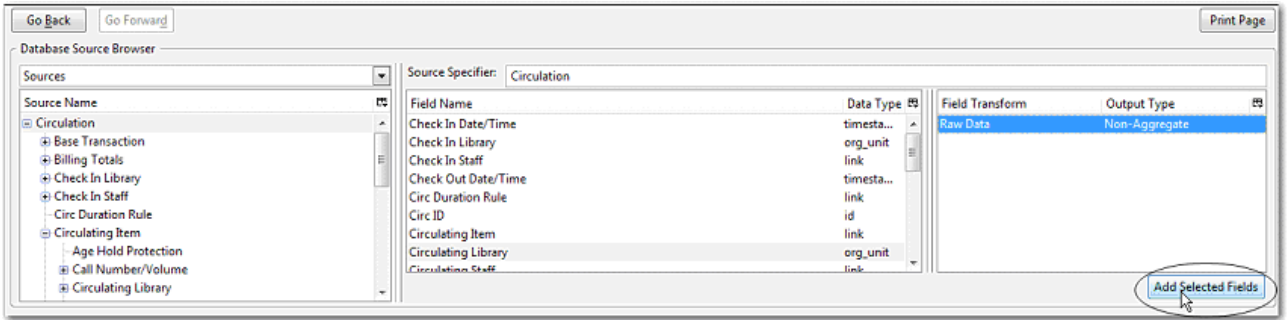
2. For this circulation statistics example, select Circulation → Check Out Date/Time → Year + Month and click on Add Selected Fields. You are going to filter on the time period.



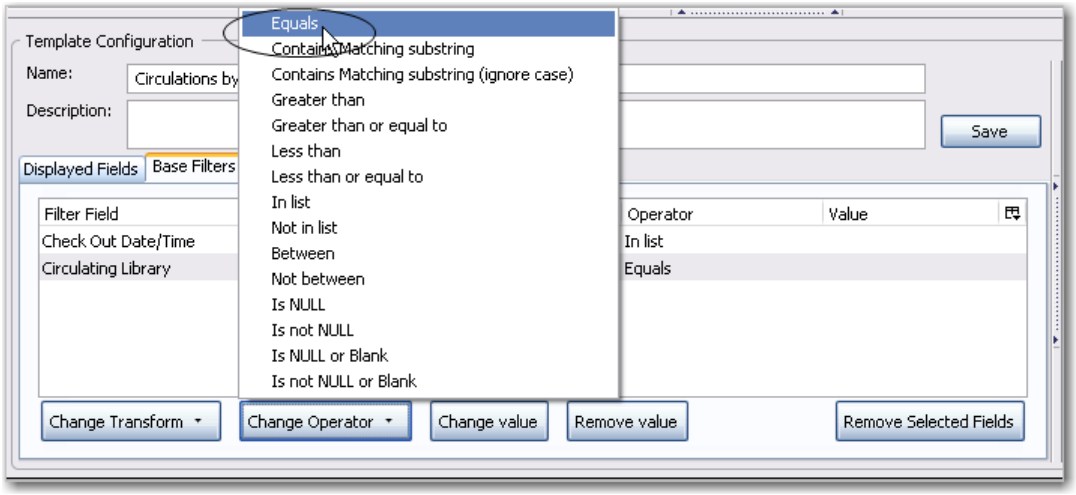
3. Select Check Out Date/Time. Click on Change Operator and select In list from the dropdown menu.



4. To filter on the location of the circulation select Circulation → Circulating library → Raw Data and click on Add Selected Fields.

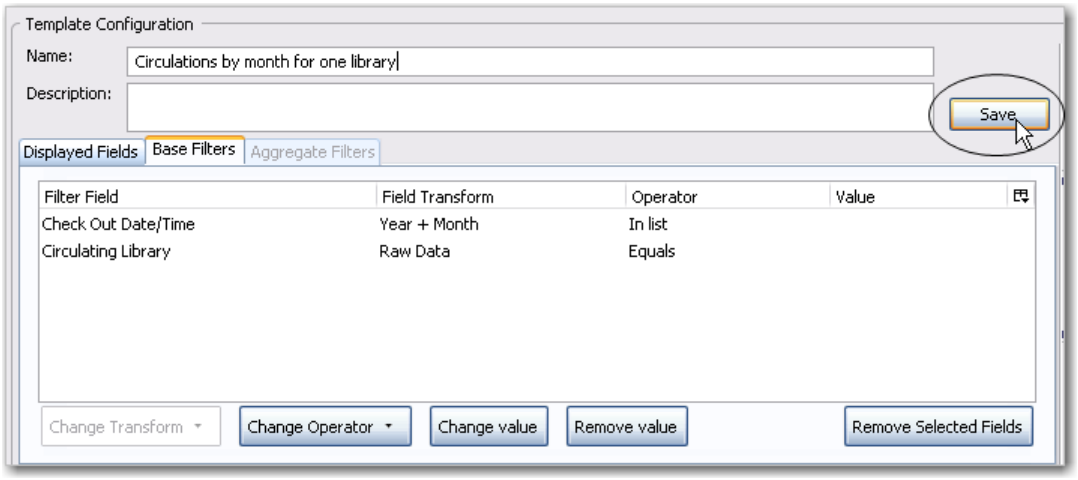


5. Select Circulating Library and click on Change Operator and select Equals. Note that this is a template, so the value for *Equals* will be filled out when you run the report

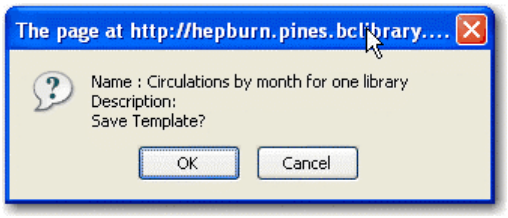


For multi-branch libraries, you would select *Circulating Library* with *In list* as the operator, so you could specify the branch(es) when you run the report. This leaves the template configurable to current requirements. In comparison, sometimes you will want to hardcode true/false values into a template. For example, deleted bibliographic records remain in the database, so perhaps you want to hardcode deleted=false, so that deleted records don't show up in the results. You might want to use deleted=true, for a template for a report on deleted items in the last month.

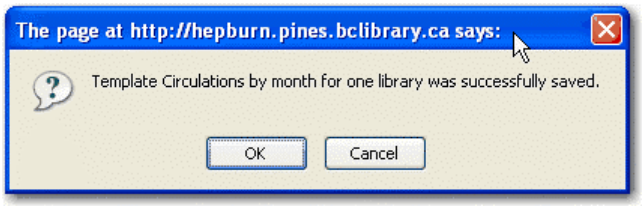
6. Once you have configured your template, you must name and save it. Name this template *Circulations by month for one library*. You can also add a description. In this example, the title is descriptive enough, so a description is not necessary. Click Save.



7. Click OK.

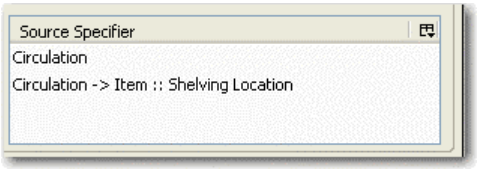


8. You will get a confirmation dialogue box that the template was successfully saved. Click OK.



After saving it is not possible to edit a template. To make changes you will need to clone it and edit the clone

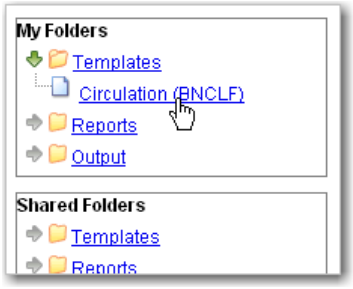
The bottom right hand pane is also a source specifier. By selecting one of these rows you will limit the fields that are visible to the sources you have specified. This may be helpful when reviewing templates with many fields. Use **Ctrl**+Click to select or deselect items.



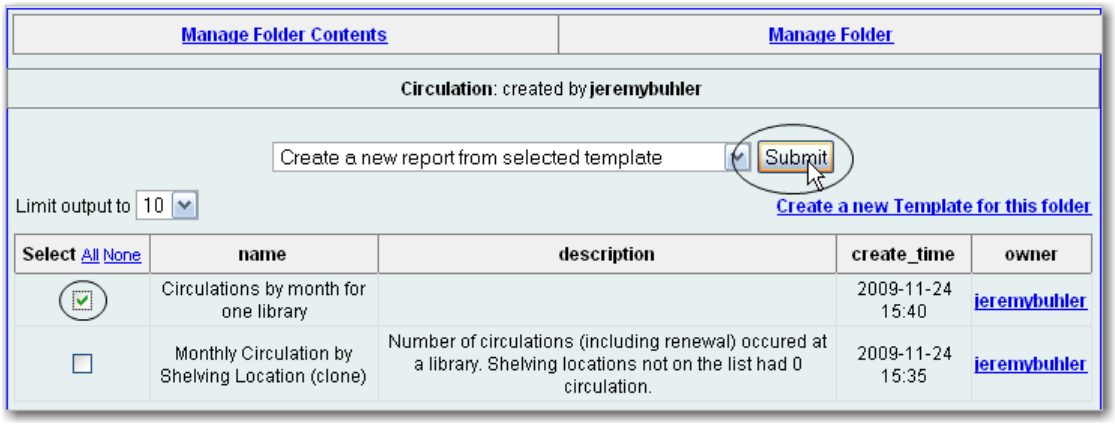
Chapter 19. Generating Reports from Templates

Now you are ready to run the report from the template you have created.

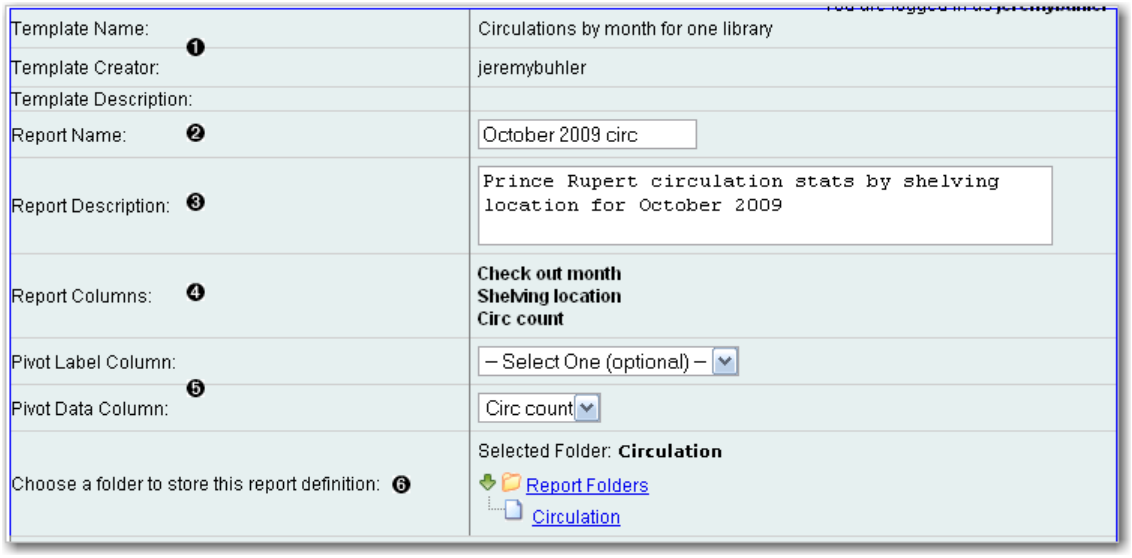
1. In the My Folders section click the arrow next to Templates to expand this folder and select circulation.



2. Select the box beside Circulations by month for one library. Select Create a new report from selected template from the dropdown menu. Click Submit.



3. Complete the first part of report settings. Only Report Name ❶ and Choose a folder... ❻ are required fields.



❶ Template Name, Template Creator, and Template Description are for informational purposes only. They are hard coded when the template is created. At the report definition stage it is not possible to change them.

❷ Report Name is required. Reports stored in the same folder must have unique names.

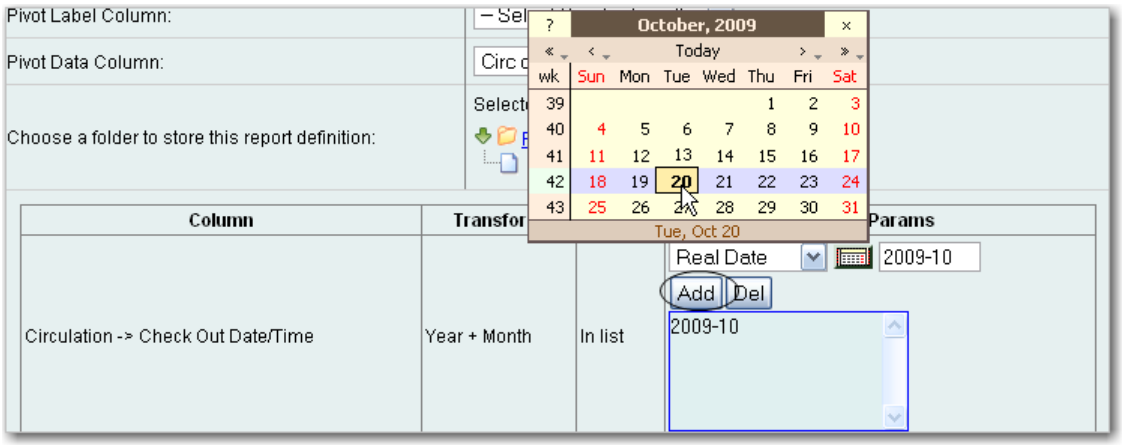
❸ Report Description is optional but may help distinguish among similar reports.

❹ Report Columns lists the columns that will appear in the output. This is derived from the template and cannot be changed during report definition.

❺ Pivot Label Column and Pivot Data Column are optional. Pivot tables are a different way to view data. If you currently use pivot tables in MS Excel it is better to select an Excel output and continue using pivot tables in Excel.

❻ You must choose a report folder to store this report definition. Only report folders under My Folders are available. Click on the desired folder to select it.

4. Select values for the Circulation > Check Out Date/Time. Use the calendar widget or manually enter the desired dates, then click Add to include the date on the list. You may add multiple dates.

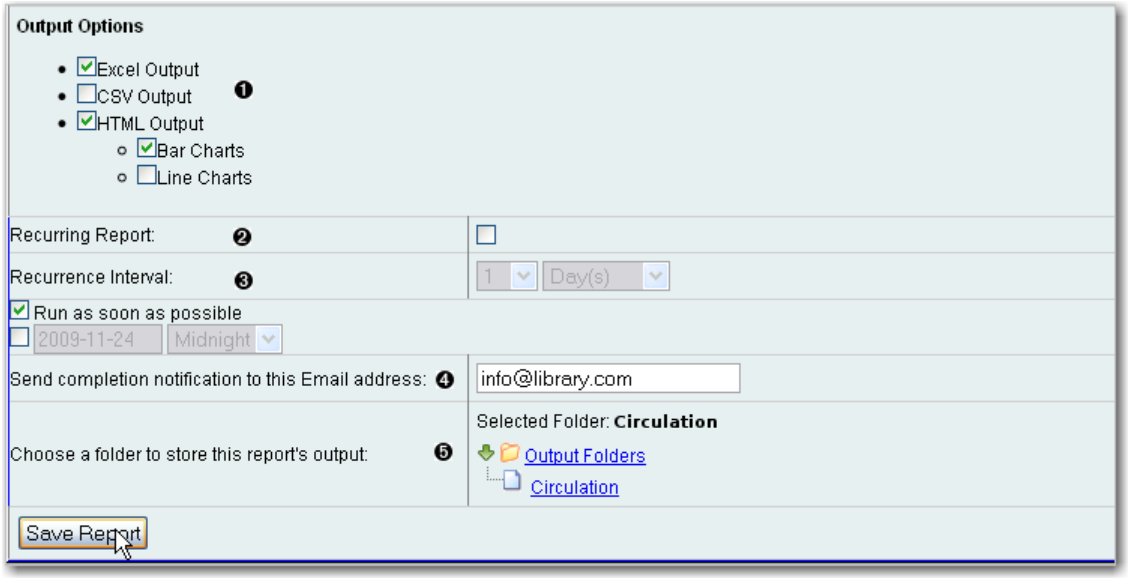


The Transform for this field is Year + Month, so even if you choose a specific date (2009-10-20) it will appear as the corresponding month only (2009-10).

It is possible to select **relative dates**. If you select a relative date *1 month ago* you can schedule reports to automatically run each month. If you want to run monthly reports that also show comparative data from one year ago, select a relative date *1 month ago*, and *13 months ago*.

5. Select a value for the Circulating Library.

6. Complete the bottom portion of the report definition interface, then click Save.



❶ Select one or more output formats. In this example the report output will be available as an Excel spreadsheet, an HTML table (for display in the staff client or browser), and as a bar chart.

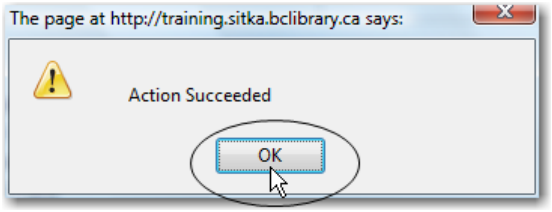
❷ If you want the report to be recurring, check the box and select the Recurrence Interval as described in Recurring Reports. In this example, as this is a report that will only be run once, the Recurring Report box is not checked.

❸ Select Run as soon as possible for immediate output. It is also possible to set up reports that run automatically at future intervals.

❹ It is optional to fill out an email address where a completion notice can be sent. The email will contain a link to password-protected report output (staff login required). If you have an email address in your Local System Administrator account it will automatically appear in the email notification box. However, you can enter a different email address or multiple addresses separated by commas.

❺ Select a folder for the report's output.

7. You will get a confirmation dialogue box that the Action Succeeded. Click OK.



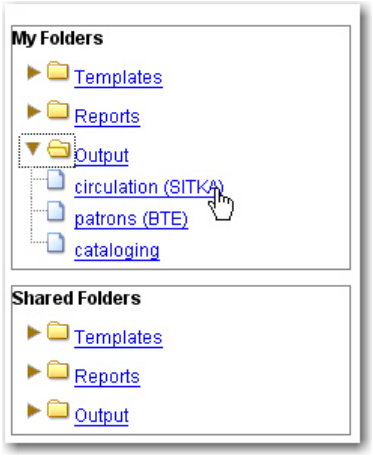
Once saved, reports stay there forever unless you delete them.

Chapter 20. Viewing Report Output

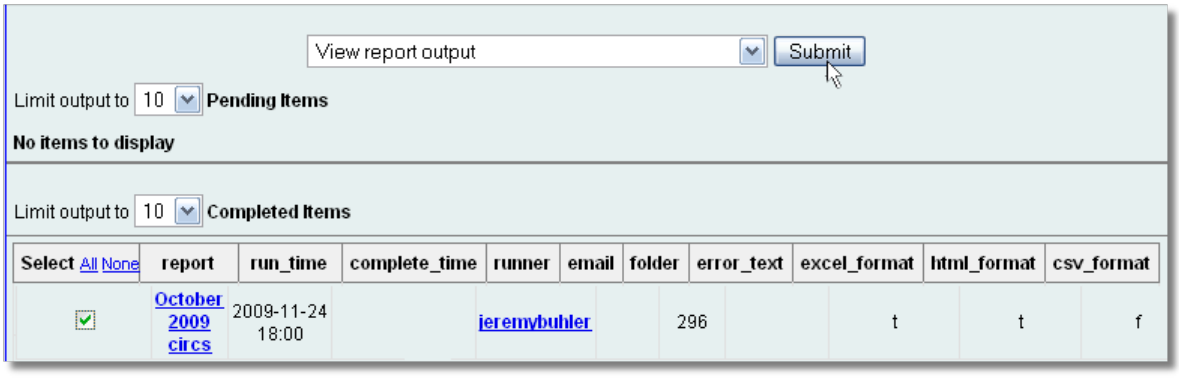
When a report runs Evergreen sends an email with a link to the output to the address defined in the report. Output is also stored in the specified Output folder and will remain there until manually deleted.

1. To view report output in the staff client, open the reports interface from Admin (-) → Local Administration → Reports

2. Click on Output to expand the folder. Select Circulation (where you just saved the *circulation report output*).



3. View report output is the default selection in the dropdown menu. Select Recurring Monthly Circ by Location by clicking the checkbox and click Submit.

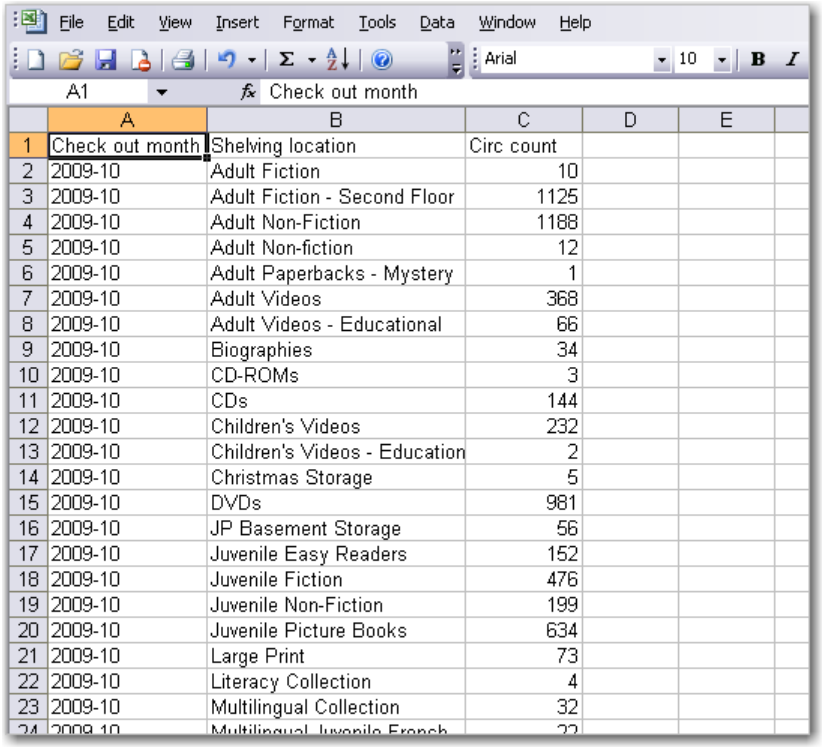


4. A new tab will open for the report output. Select either Tabular Output or Excel Output. If Bar Charts was selected during report definition the chart will also appear.

5. Tabular output looks like this:

Check out month	Shelving location	Circ count
2009-10	Adult Fiction	10
2009-10	Adult Fiction - Second Floor	1125
2009-10	Adult Non-Fiction	1188
2009-10	Adult Non-fiction	12
2009-10	Adult Paperbacks - Mystery	1
2009-10	Adult Videos	368
2009-10	Adult Videos - Educational	66
2009-10	Biographies	34
2009-10	CD-ROMs	3
2009-10	CDs	144
2009-10	Children's Videos	232
2009-10	Children's Videos - Educational	2
2009-10	Christmas Storage	5
2009-10	DVDs	981
2009-10	JP Basement Storage	56
2009-10	Juvenile Easy Readers	152
2009-10	Juvenile Fiction	476
2009-10	Juvenile Non-Fiction	199
2009-10	Juvenile Picture Books	634
2009-10	Large Print	73

6. If you want to manipulate, filter or graph this data, Excel output would be more useful. Excel output looks like this in Excel:



	A	B	C	D	E
1	Check out month	Shelving location	Circ count		
2	2009-10	Adult Fiction	10		
3	2009-10	Adult Fiction - Second Floor	1125		
4	2009-10	Adult Non-Fiction	1188		
5	2009-10	Adult Non-fiction	12		
6	2009-10	Adult Paperbacks - Mystery	1		
7	2009-10	Adult Videos	368		
8	2009-10	Adult Videos - Educational	66		
9	2009-10	Biographies	34		
10	2009-10	CD-ROMs	3		
11	2009-10	CDs	144		
12	2009-10	Children's Videos	232		
13	2009-10	Children's Videos - Education	2		
14	2009-10	Christmas Storage	5		
15	2009-10	DVDs	981		
16	2009-10	JP Basement Storage	56		
17	2009-10	Juvenile Easy Readers	152		
18	2009-10	Juvenile Fiction	476		
19	2009-10	Juvenile Non-Fiction	199		
20	2009-10	Juvenile Picture Books	634		
21	2009-10	Large Print	73		
22	2009-10	Literacy Collection	4		
23	2009-10	Multilingual Collection	32		
24	2009-10	Multilingual Juvenile French	22		

Chapter 21. Cloning Shared Templates

This chapter describes how to make local copies of shared templates for routine reports or as a starting point for customization. When creating a new template it is a good idea to review the shared templates first: even if the exact template you need does not exist it is often faster to modify an existing template than to build a brand new one. A Local System Administrator account is required to clone templates from the Shared Folders section and save them to My Folders.

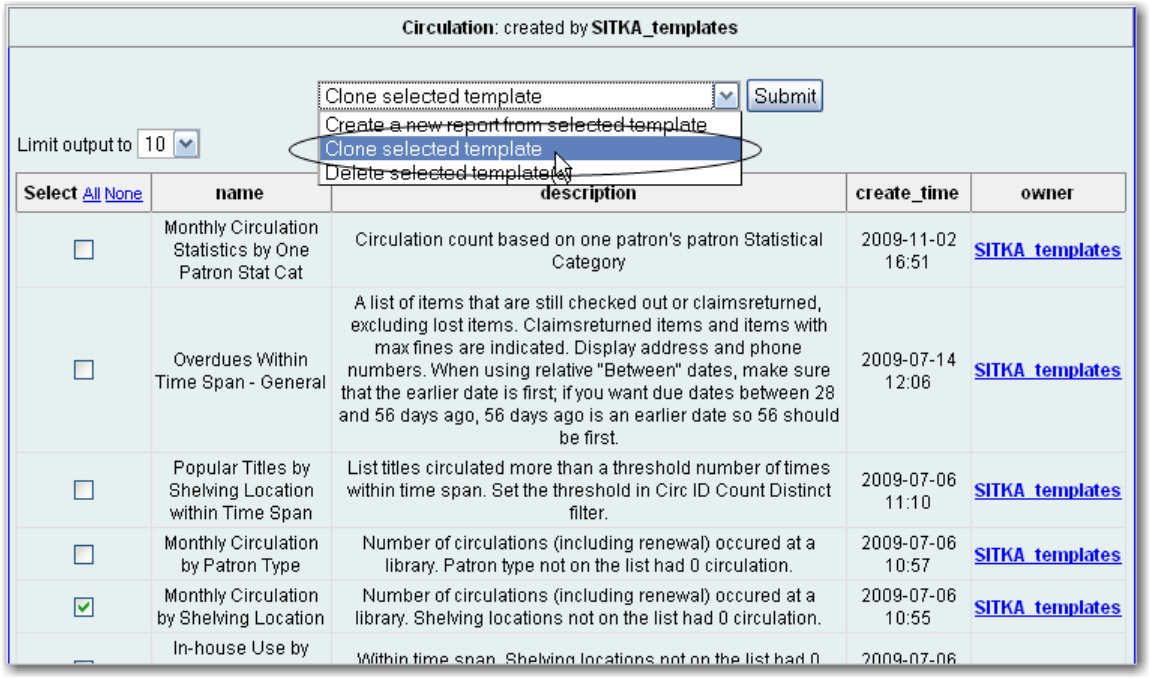
The steps below assume you have already created at least one Templates folder. If you haven't done this, please see Chapter 17, *Folders*.

1. Access the reports interface from the Admin (-) menu under Local Administration → Reports

2. Under Shared Folders expand the Templates folder and the subfolder of the report you wish to clone. To expand the folders click on the grey arrow or folder icon. Do not click on the blue underlined hyperlink.

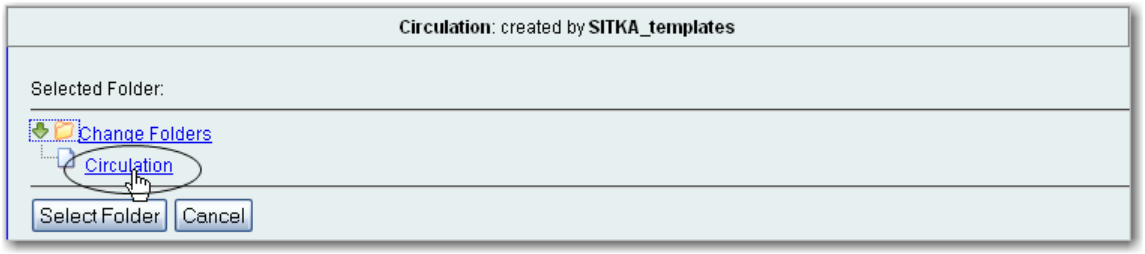
3. Click on the subfolder.

4. Select the template you wish to clone. From the dropdown menu choose Clone selected templates, then click Submit.

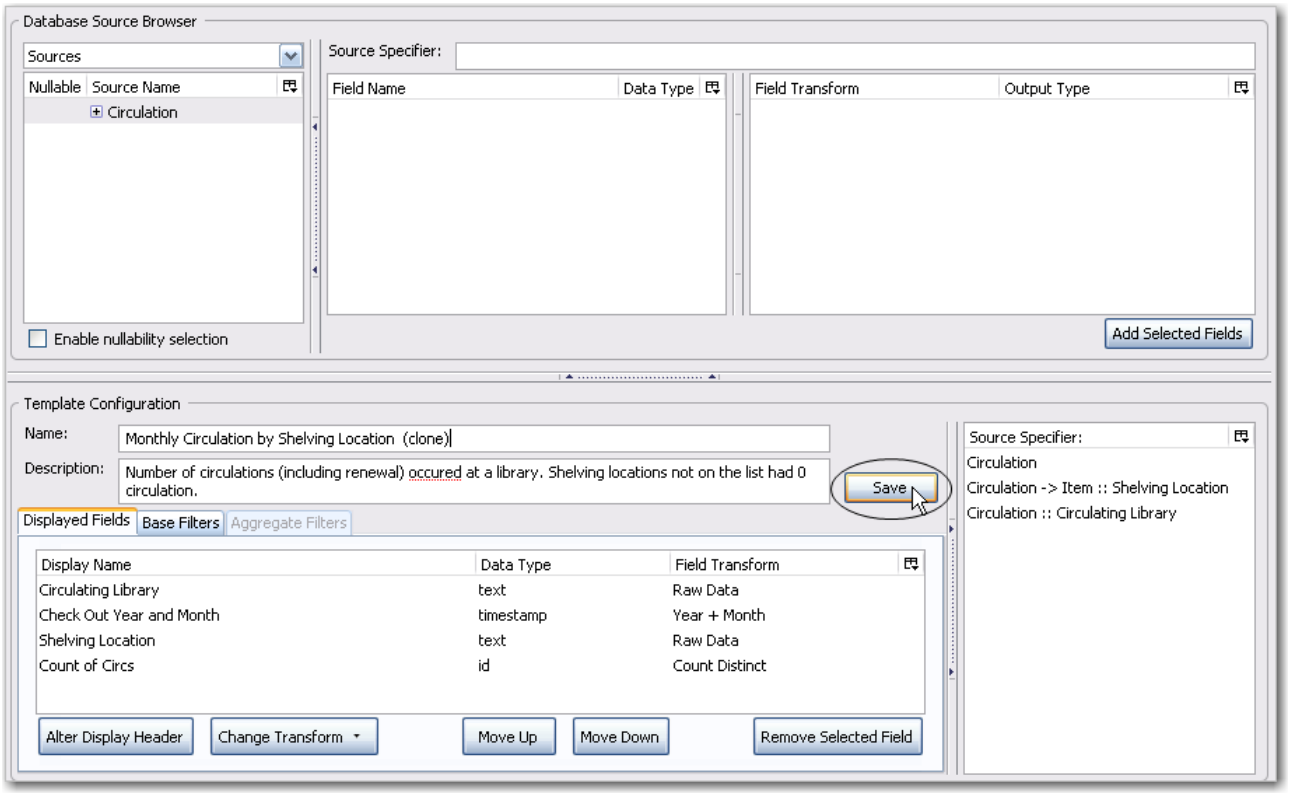


By default Evergreen only displays the first 10 items in any folder. To view all content, change the Limit output setting from 10 to All.

5. Choose the folder where you want to save the cloned template, then click Select Folder. Only template folders created with your account will be visible. If there are no folders to choose from please see Chapter 17, *Folders*.



6. The cloned template opens in the template editor. From here you may modify the template by adding, removing, or editing fields and filters as described in Chapter 18, *Creating Templates*. Template Name and Description can also be edited. When satisfied with your changes click Save.



7. Click OK in the resulting confirmation windows.

Once saved it is not possible to edit a template. To make changes, clone a template and change the clone.

Chapter 22. Adding Data Sources to Reporter

You can further customize your Evergreen reporting environment by adding additional data sources.

The Evergreen reporter module does not build and execute SQL queries directly, but instead uses a data abstraction layer called Fieldmapper to mediate queries on the Evergreen database. Fieldmapper is also used by other core Evergreen DAO services, including cstore and permacrud. The configuration file fm_IDL.xml contains the mapping between Fieldmapper class definitions and the database. The fm_IDL.xml file is located in the /openils/conf directory.

There are 3 basic steps to adding a new data source. Each step will be discussed in more detail in the

1. Create a PostgreSQL query, view, or table that will provide the data for your data source.

2. Add a new class to fm_IDL.xml for your data source.

3. Restart the affected services to see the new data source in Reporter.

There are two possbile sources for new data sources:

- An SQL query built directly into the class definition in fm_IDL.xml. You can use this method if you are only going to access this data source through the Evergreen reporter and/or cstore code that you write.
- A new table or view in the Evergreen PostgresSQL database on which a class definition in fm_IDL.xml. You can use this method if you want to be able to access this data source through directly through SQL or using other reporting tool.

Create a PostgreSQL query, view, or table that will provide the data for your data source

You need to decide whether you will create your data source as a query, a view, or a table.

- Create a *query* if you are planning to access this data source only through the Evergreen reporter and/or cstore code that you write. You will use this query to create an IDL only view.
- Create a *view* if you are planning to access this data source through other methods in addition to the Evergreen reporter, or if you may need to do performance tuning to optimize your query.
- You may also need to use an additional *table* as part of your data source if you have additional data that's not included in the base Evergreen, or if you need to use a table to store the results of a query for performance reasons.

To develop and test queries, views, and tables, you will need

- Access to the Evergree PostgreSQL database at the command line. This is normally the psql application. For introductory information, please see ???. You can access the Postgres documentation at the Official Postgres documentation for more information about PostgreSQL.
- Knowledge of the Evergreen database structure for the data that you want to access. You can find this information by looking at the Evergreen schema - see Chapter 26, *Database Schema* and ???

If the views that you are creating are purely local in usage are are intended for contribution to the core Evergreen code, create the Views and Tables in the extend_reporter schema. This schema is intended to be used for local customizations and will not be modified during upgrades to the Evergreen system.

You should make that you have an appropriate version control pocess for the SQL used to create you data sources.

Here's an example of a view created to incorporate some locally defined user statistical categories.

```
create view extend_reporter.patronstats as
select u.id,
grp.name as "ptype",
rl.stat_cat_entry as "reg_lib",
gr.stat_cat_entry as "gender",
ag.stat_cat_entry as "age_group",
EXTRACT(YEAR FROM age(u.dob)) as "age",
hl.id as "home_lib",
u.create_date,
u.expire_date,
ms_balance_owed
from actor.usr u
join permission.grp_tree grp on (u.profile = grp.id and (grp.parent = 2 or grp.name = 'patron'))
join actor.org_unit hl on (u.home_ou = hl.id)
left join money.open_usr_summary ms on (ms.usr = u.id)
left join actor.stat_cat_entry_usr_map rl on (u.id = rl.target_usr and rl.stat_cat = 4)
left join actor.stat_cat_entry_usr_map bt on (u.id = bt.target_usr and bt.stat_cat = 3)
left join actor.stat_cat_entry_usr_map gr on (u.id = gr.target_usr and gr.stat_cat = 2)
left join actor.stat_cat_entry_usr_map gr on (u.id = gr.target_usr and gr.stat_cat = 2)
left join actor.stat_cat_entry_usr_map ag on (u.id = ag.target_usr and ag.stat_cat = 1)
where u.active = 't' and u.deleted <> 't';
```

Add a new class to fm_IDL.xml for your data source

Once you have your data source, the next step is to add that data source as a new class in fm_IDL.xml.

You will need to add the following attributes for the class definition

- *id*. You should follow a consistent naming convention for your class names that won't create conflicts in the future with any standard classes added in future upgrades. Evergreen normally names each class with the first letter of each word in the schema and table names. You may want to add a local prefix or suffix to your local class names.
- *controller=”open-ils.cstore”*
- *oils_obj:fieldmapper*=”extend_reporter::long_name_of_view”
- *oils_persist.readonly*=”true”
- *reporter:core*=”true” (if you want this to show up as a “core” reporting source)
- *reporter*:label. This is the name that will appear on the data source list in the Evergreen reporter.
- *oils_persist:source_definition*. If this is an IDL-only view, add the SQL query here. You don't need this attribute if your class is based on a PostgreSQL view or table.
- *oils_persist:tablename*="schemaname.viewname or tablename" If this class is based on a PostgreSQL view or table, add the table name here. You don't need this attribute is your class is an IDL-only view.

For each column in the view or query output, add *field* element and set the following attributes. The fields should be wrapped with <field> </field>

- reporter:label. This is the name that appears in the Evergreen reporter.
- name. This should match the column name in the view or query output.
- reporter:datatype (which can be id, bool, money, org_unit, int, number, interval, float, text, timestamp, or link)

For each linking field, add a *link* element with the following attributes. The elements should be wrapped with <link> </link>

- field (should match field.name)
- reltype (“has_a”, “might_have”, or “has_many”)
- map (“”)
- key (name of the linking field in the foreign table)
- class (ID of the IDL class of the table that is to be linked to)

The following example is a class definition for the example view that was created in the previous section.

```
<class id="erpstats" controller="open-ils.reporter-store" oils_obj:fieldmapper="extend_reporter::patronstats"
oils_persist:tablename="extend_reporter.patronstats" oils_persist:readonly="true" reporter:label="Patron Statistics"
  <fields oils_persist:primary="id">
  <field reporter:label="Patron ID" name="id" reporter:datatype="link" />
  <field reporter:label="Patron Type" name="ptype" reporter:datatype="text" />
  <field reporter:label="Reg Lib" name="reg_lib" reporter:datatype="text" />
  <field reporter:label="Boro/Twp" name="boro_twp" reporter:datatype="text" />
  <field reporter:label="Gender" name="gender" reporter:datatype="text" />
  <field reporter:label="Age Group" name="age_group" reporter:datatype="text" />
  <field reporter:label="Age" name="age" reporter:datatype="int" />
  <field reporter:label="Home Lib ID" name="home_lib_id" reporter:datatype="link" />
  <field reporter:label="Home Lib Code" name="home_lib_code" reporter:datatype="text" />
  <field reporter:label="Home Lib" name="home_lib" reporter:datatype="text" />
  <field reporter:label="Create Date" name="create_date" reporter:datatype="timestamp" />
  <field reporter:label="Expire Date" name="expire_date" reporter:datatype="timestamp" />
  <field reporter:label="Balance Owed" name="balance_owed" reporter:datatype="money" />
</fields>
<links>
  <link field="id" reltype="has_a" key="id" map="" class="au"/>
  <link field="home_lib_id" reltype="has_a" key="id" map="" class="aou"/>
</links>
</class>
```

fm_IDL.xml is used by other core Evergreen DAO services, including cstore and permacrud. So changes to this file can affect the entire Evergreen application, not just reporter. After making changes fm_IDL.xml, it is a good idea to ensure that it is valid XML by using a utility such as xmllint – a syntax error can render much of Evergreen nonfunctional. Set up a good change control system for any changes to fm_IDL.xml. You will need to keep a separate copy of you local class definitions so that you can reapply the changes to fm_IDL.xml after Evergreen upgrades.

Restart the affected services to see the new data source in the reporter

The following steps are needed to for Evergreen to recognize the changes to fm_IDL.xml

1. Copy the updated fm_IDL.xml Update /openils/conf/fm_IDL.xml to /openils/var/web/reports/fm_IDL.xml

   ```
   cp /openils/conf/fm_IDL.xml /openils/var/web/reports/fm_IDL.xml
   ```

2. Run Autogen to to update the Javascript versions of the fieldmapper definitions.

   ```
   /openils/bin/autogen.sh
   ```

3. Restart C services

   ```
   osrf_ctl.sh -l -a restart_c
   ```

4. Restart the Evergreen reporter. You may need to modify this command depending on your system configuration and pid path

   ```
   opensrf-perl.pl -l -action restart -service open-ils.reporter \
   -config /openils/conf/opensrf_core.xml -pid-dir /openils/var/run
   ```

5. Restart the Evergreen application or use Admin, For Developers, Clear Cache

Chapter 23. Running Recurring Reports

Recurring reports are a useful way to save time by scheduling reports that you run on a regular basis, such as monthly circulation and monthly patron registration statistics. When you have set up a report to run on a monthly basis you'll get an email informing you that the report has successfully run. You can click on a link in the email that will take you directly to the report output. You can also access the output through the reporter interface as described in Chapter 20, *Viewing Report Output*.

To set up a monthly recurring report follow the procedure in Generating Reports from Templates but make the changes described below.

1. Select the Recurring Report check-box and set the recurrence interval to 1 month.
2. Do not select Run ASAP. Instead schedule the report to run early on the first day of the next month. Enter the date in YYYY-MM-DD format.
3. Ensure there is an email address to receive completion emails. You will receive an email completion notice each month when the output is ready.
4. Select a folder for the report's output.
5. Click Save Report.
6. You will get a confirmation dialogue box that the Action Succeeded. Click OK.

You will get an email on the 1st of each month with a link to the report output. By clicking this link it will open the output in a web browser. It is still possible to login to the staff client and access the output in Output folder.

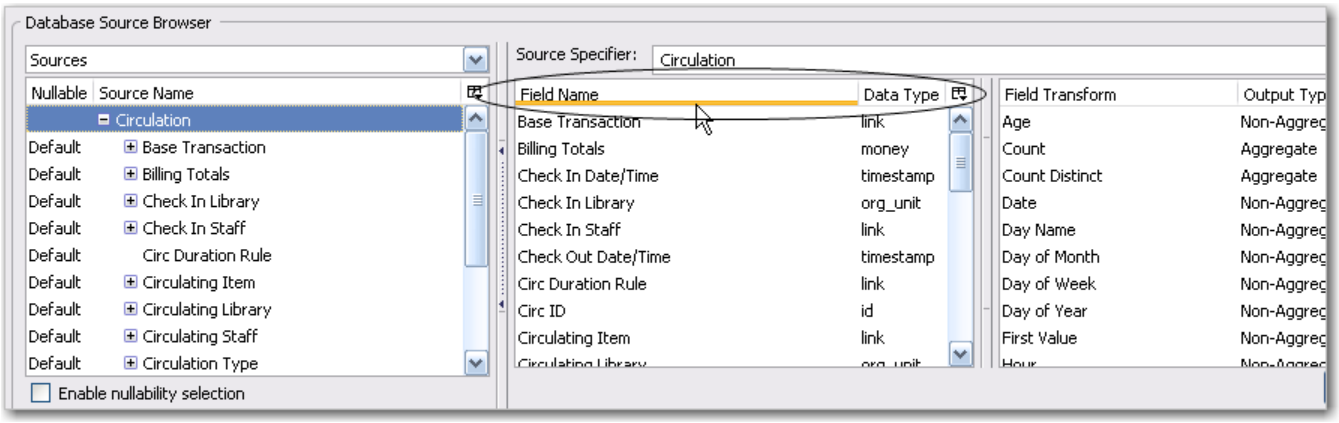
How to stop or make changes to an existing recurring report? Sometimes you may wish to stop or make changes to a recurring report, e.g. the recurrence interval, generation date, email address to receive completion email, output format/folder or even filter values (such as the number of days overdue). You will need to delete the current report from the report folder, then use the above procedure to set up a new recurring report with the desired changes. Please note that deleting a report also deletes all output associated with it.

Once you have been on Evergreen for a year, you could set up your recurring monthly reports to show comparative data from one year ago. To do this select relative dates of 1 month ago and 13 months ago.

Chapter 24. Template Terminology

Data Types

The central column of the Database Source Browser lists Field Name and Data Type for the selected database table.

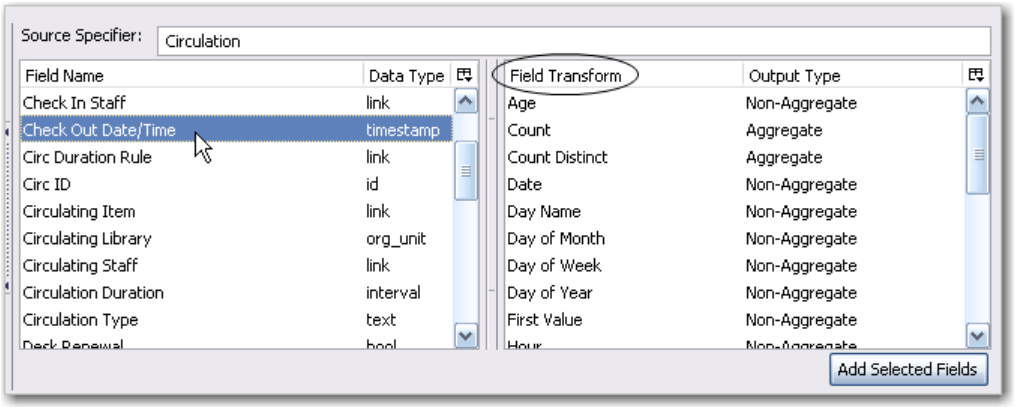


Each data type has its own characteristics and uses:

Data Type	Description	Notes
id	Unique number assigned by the database to identify a record	A number that is a meaningful reference for the database but not of much use to a human user. Use in displayed fields when counting records or in filters.
text	Text field	Usually uses the Raw Data transform.
timestamp	Exact date and time	Select appropriate date/time transform. Raw Data includes second and timezone information, usually more than is required for a report.
bool	True or False	Commonly used to filter out deleted item or patron records.
org_unit	A number representing a library, library system, or federation	When you want to filter on a library, make sure that the field name is on an org_unit or id data type.
link	A link to another database table	Link outputs a number that is a meaningful reference for the database but not of much use to a human user. You will usually want to drill further down the tree in the Sources pane and select fields from the linked table. However, in some instances you might want to use a link field. For example, to count the number of patrons who borrowed items you could do a count on the Patron link data.
int	Integer	
money	Number (in dollars)	

Field Transforms

A Field Transform tells the reporter how to process a field for output. Different data types have different transform options.



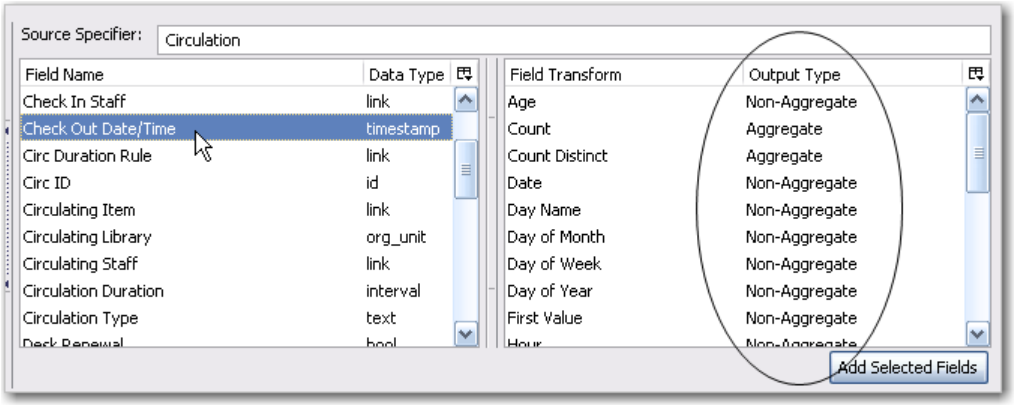
Raw Data. To display a field exactly as it appears in the database use the Raw Data transform, available for all data types.

Count and Count Distinct. These transforms apply to the id data type and are used to count database records (e.g. for circulation statistics). Use Count to tally the total number of records. Use Count Distinct to count the number of unique records, removing duplicates.

To demonstrate the difference between Count and Count Distinct, consider an example where you want to know the number of active patrons in a given month, where *active* means they borrowed at least one item. Each circulation is linked to a Patron ID, a number identifying the patron who borrowed the item. If we use the Count Distinct transform for Patron IDs we will know the number of unique patrons who circulated at least one book (2 patrons in the table below). If instead, we use Count, we will know how many books were circulated, since every circulation is linked to a patron ID and duplicate values are also counted. To identify the number of active patrons in this example the Count Distinct transform should be used.

Title	**Patron ID**	**Patron Name**
Harry Potter and the Chamber of Secrets	001	John Doe
Northern Lights	001	John Doe
Harry Potter and the Philosopher's Stone	222	Jane Doe

Output Type. Note that each transform has either an Aggregate or Non-Aggregate output type.



Selecting a Non-Aggregate output type will return one row of output in your report for each row in the database. Selecting an Aggregate output type will group together several rows of the database and return just one row of output with, say, the average value or the total count for that group. Other common aggregate types include minimum, maximum, and sum.

When used as filters, non-aggregate and aggregate types correspond to Base and Aggregate filters respectively. To see the difference between a base filter and an aggregate filter, imagine that you are creating a report to count the number of circulations in January. This would require a base filter to specify the month of interest because the month is a non-aggregate output type. Now imagine that you wish to list all items with more than 25 holds. This would require an aggregate filter on the number of holds per item because you must use an aggregate output type to count the holds.

Chapter 25. Exporting Report Templates Using phpPgAdmin

Once the data is exported. Database Administrators/Systems Administrators can easily import this data into the templates folder to make it available in the client.

Dump the Entire Reports Template Table

The data exported in this method can create issues importing into a different system if you do not have a matching folder and owner. This is going to export report templates created in your system. The most important fields for importing into the new system are *name*, *description*, and *data*. Data defines the actual structure of the report. The *owner* and *folder* fields will unique to the system they were exported from and will have to be altered to ensure they match the appropriate owner and folder information for the new system.

1. Go to the **Reporter** schema. Report templates are located in the **Template** table
2. Click on the link to the **Template** table
3. Click the **export** button at the top right of the phpPgAdmin screen
4. Make sure the following is selected
    a. Data Only (checked)
    b. Format: Select CSV or Tabbed did get the data in a text format
    c. Download checked
5. Click export button at the bottom
6. A text file will download to your local system

Dump Data with an SQL Statement

The following statement could be used to grab the data in the folder and dump it with admin account as the owner and the first folder in your system.

```
SELECT 1 as owner, name, description, data, 1 as folder FROM reporter.template
```

or use the following to capture your folder names for export

```
SELECT 1 as owner, t.name, t.description, t.data, f.name as folder
 FROM reporter.template t
 JOIN reporter.template_folder f ON t.folder=f.id
```

1. Run the above query
2. Click the **download** link at the bottom of the page
3. Select the file format (CSV or Tabbed)

4. Check download

5. A text file with the report template data will be downloaded.

Part VI. Third Party System Integration

Part VII. Development

Part VIII. Appendices

Table of Contents

Appendix A. Permissions List

Permission Descriptions

ABORT_REMOTE_TRANSIT. Allow a user to abort a copy transit if the user is not at the transit source or dest

ABORT_TRANSIT. Allow a user to abort a copy transit if the user is at the transit destination or source

ACQ_XFER_MANUAL_DFUND_AMOUNT. Allow a user to transfer different amounts of money out of one fund and into another

actor.org_unit.closed_date.create. Allow a user to create a new closed date for a location

actor.org_unit.closed_date.delete. Allow a user to remove a closed date interval for a given location

actor.org_unit.closed_date.update. Allow a user to update a closed date interval for a given location

ADMIN_ACQ_CANCEL_CAUSE. Allow a user to create/update/delete reasons for order cancellations

ADMIN_ACQ_CLAIM. ADMIN_ACQ_CLAIM

ADMIN_ACQ_CLAIM_EVENT_TYPE. ADMIN_ACQ_CLAIM_EVENT_TYPE

ADMIN_ACQ_CLAIM_TYPE. ADMIN_ACQ_CLAIM_TYPE

ADMIN_ACQ_DISTRIB_FORMULA. ADMIN_ACQ_DISTRIB_FORMULA

ADMIN_ACQ_FISCAL_YEAR. ADMIN_ACQ_FISCAL_YEAR

ADMIN_ACQ_FUND. Allow a user to create/view/update/delete a fund

ADMIN_ACQ_FUND_ALLOCATION_PERCENT. ADMIN_ACQ_FUND_ALLOCATION_PERCENT

ADMIN_ACQ_FUND_TAG. ADMIN_ACQ_FUND_TAG

ADMIN_ACQ_LINEITEM_ALERT_TEXT. ADMIN_ACQ_LINEITEM_ALERT_TEXT

ADMIN_AGE_PROTECT_RULE. ADMIN_AGE_PROTECT_RULE

ADMIN_ASSET_COPY_TEMPLATE. ADMIN_ASSET_COPY_TEMPLATE

ADMIN_BOOKING_RESERVATION. Enables the user to create/update/delete booking reservations

ADMIN_BOOKING_RESERVATION_ATTR_MAP.
ADMIN_BOOKING_RESERVATION_ATTR_MAP

ADMIN_BOOKING_RESERVATION_ATTR_VALUE_MAP. Enables the user to create/update/delete booking reservation attribute value maps

ADMIN_BOOKING_RESOURCE. Enables the user to create/update/delete booking resources

ADMIN_BOOKING_RESOURCE_ATTR. Enables the user to create/update/delete booking resource attributes

ADMIN_BOOKING_RESOURCE_ATTR_MAP. Enables the user to create/update/delete booking resource attribute maps

ADMIN_BOOKING_RESOURCE_ATTR_VALUE. Enables the user to create/update/delete booking resource attribute values

ADMIN_BOOKING_RESOURCE_TYPE. Enables the user to create/update/delete booking resource types

ADMIN_CIRC_MATRIX_MATCHPOINT. ADMIN_CIRC_MATRIX_MATCHPOINT

ADMIN_CIRC_MOD. ADMIN_CIRC_MOD

ADMIN_CLAIM_POLICY. ADMIN_CLAIM_POLICY

ADMIN_CONFIG_REMOTE_ACCOUNT. ADMIN_CONFIG_REMOTE_ACCOUNT

ADMIN_COPY_LOCATION_ORDER. Allow a user to create/view/update/delete a copy location order

ADMIN_CREDIT_CARD_PROCESSING. Update org unit settings related to credit card processing

ADMIN_CURRENCY_TYPE. Allow a user to create/view/update/delete a currency_type

ADMIN_FIELD_DOC. ADMIN_FIELD_DOC

ADMIN_FUND. Allow a user to create/view/update/delete a fund

ADMIN_FUNDING_SOURCE. Allow a user to create/view/update/delete a funding source

ADMIN_GLOBAL_FLAG. ADMIN_GLOBAL_FLAG

ADMIN_GROUP_PENALTY_THRESHOLD. ADMIN_GROUP_PENALTY_THRESHOLD

ADMIN_HOLD_CANCEL_CAUSE. ADMIN_HOLD_CANCEL_CAUSE

ADMIN_HOLD_MATRIX_MATCHPOINT. ADMIN_HOLD_MATRIX_MATCHPOINT

ADMIN_IDENT_TYPE. ADMIN_IDENT_TYPE

ADMIN_IMPORT_ITEM_ATTR_DEF. ADMIN_IMPORT_ITEM_ATTR_DEF

ADMIN_INDEX_NORMALIZER. ADMIN_INDEX_NORMALIZER

ADMIN_INVOICE. ADMIN_INVOICE

ADMIN_INVOICE_METHOD. ADMIN_INVOICE_METHOD

ADMIN_INVOICE_PAYMENT_METHOD. ADMIN_INVOICE_PAYMENT_METHOD

ADMIN_LINEITEM_MARC_ATTR_DEF. ADMIN_LINEITEM_MARC_ATTR_DEF

ADMIN_MARC_CODE. ADMIN_MARC_CODE

ADMIN_MAX_FINE_RULE. ADMIN_MAX_FINE_RULE

ADMIN_MERGE_PROFILE. ADMIN_MERGE_PROFILE

ADMIN_ORG_UNIT_SETTING_TYPE. ADMIN_ORG_UNIT_SETTING_TYPE

ADMIN_PROVIDER. Allow a user to create/view/update/delete a provider

ADMIN_RECURRING_FINE_RULE. ADMIN_RECURRING_FINE_RULE

ADMIN_SERIAL_CAPTION_PATTERN. Create/update/delete serial caption and pattern objects

ADMIN_SERIAL_DISTRIBUTION. Create/update/delete serial distribution objects

ADMIN_SERIAL_STREAM. Create/update/delete serial stream objects

ADMIN_SERIAL_SUBSCRIPTION. Create/update/delete serial subscription objects

ADMIN_STANDING_PENALTY. ADMIN_STANDING_PENALTY

ADMIN_SURVEY. ADMIN_SURVEY

ADMIN_TRIGGER_CLEANUP. Allow a user to create, delete, and update trigger cleanup entries

ADMIN_TRIGGER_EVENT_DEF. Allow a user to administer trigger event definitions

ADMIN_TRIGGER_HOOK. Allow a user to create, update, and delete trigger hooks

ADMIN_TRIGGER_REACTOR. Allow a user to create, update, and delete trigger reactors

ADMIN_TRIGGER_TEMPLATE_OUTPUT. Allow a user to delete trigger template output

ADMIN_TRIGGER_VALIDATOR. Allow a user to create, update, and delete trigger validators

ADMIN_USER_REQUEST_TYPE. ADMIN_USER_REQUEST_TYPE

ADMIN_USER_SETTING_GROUP. ADMIN_USER_SETTING_GROUP

ADMIN_USER_SETTING_TYPE. ADMIN_USER_SETTING_TYPE

ADMIN_Z3950_SOURCE. ADMIN_Z3950_SOURCE

ALLOW_ALT_TCN. Allows staff to import a record using an alternate TCN to avoid conflicts

ASSIGN_GROUP_PERM. ASSIGN_GROUP_PERM

ASSIGN_WORK_ORG_UNIT. Allow a staff member to define where another staff member has their permissions

BAR_PATRON. Allow a user to bar a patron

CANCEL_HOLDS. Allow a user to cancel holds

CAPTURE_RESERVATION. Allows a user to capture booking reservations

CHECKIN_BYPASS_HOLD_FULFILL. * no longer applicable

CIRC_CLAIMS_RETURNED.override. Allow a user to check in or check out an item that has a status of 'claims returned'

CIRC_EXCEEDS_COPY_RANGE.override. Allow staff to override circulation copy range failure

CIRC_OVERRIDE_DUE_DATE. Allow a user to change the due date on an item to any date

CIRC_PERMIT_OVERRIDE. Allow a user to bypass the circulation permit call for check out

COPY_ALERT_MESSAGE.override. Allow a user to check in/out an item that has an alert message

COPY_BAD_STATUS.override. Allow a user to check out an item in a non-circulatable status

COPY_CHECKIN. Allow a user to check in a copy

COPY_CHECKOUT. Allow a user to check out a copy

COPY_CIRC_NOT_ALLOWED.override. Allow a user to checkout an item that is marked as non-circ

COPY_HOLDS. Allow a user to place a hold on a specific copy

COPY_IS_REFERENCE.override. Allow a user to override the copy_is_reference event

COPY_NEEDED_FOR_HOLD.override. Allow a user to force renewal of an item that could fulfill a hold request

COPY_NOT_AVAILABLE.override. Allow staff to force checkout of Missing/Lost type items

COPY_STATUS_LOST.override. Allow a user to remove the lost status from a copy

COPY_STATUS_MISSING.override. Allow a user to change the missing status on a copy

COPY_TRANSIT_RECEIVE. Allow a user to close out a transit on a copy

CREATE_ACQ_FUNDING_SOURCE. CREATE_ACQ_FUNDING_SOURCE

CREATE_AUDIENCE. CREATE_AUDIENCE

CREATE_AUTHORITY_IMPORT_IMPORT_FIELD_DEF.
CREATE_AUTHORITY_IMPORT_IMPORT_FIELD_DEF

CREATE_AUTHORITY_IMPORT_QUEUE. CREATE_AUTHORITY_IMPORT_QUEUE

CREATE_AUTHORITY_RECORD_NOTE. CREATE_AUTHORITY_RECORD_NOTE

CREATE_BIB_BTYPE. CREATE_BIB_BTYPE

CREATE_BIB_IMPORT_FIELD_DEF. CREATE_BIB_IMPORT_FIELD_DEF

CREATE_BIB_IMPORT_QUEUE. CREATE_BIB_IMPORT_QUEUE

CREATE_BIB_LEVEL. CREATE_BIB_LEVEL

CREATE_BIBLIO_FINGERPRINT. CREATE_BIBLIO_FINGERPRINT

CREATE_BIB_SOURCE. CREATE_BIB_SOURCE

CREATE_BILL. Allow a user to create a new bill on a transaction

CREATE_BILLING_TYPE. CREATE_BILLING_TYPE

CREATE_CIRC_DURATION. CREATE_CIRC_DURATION

CREATE_CIRC_MOD. CREATE_CIRC_MOD

CREATE_CN_BTYPE. CREATE_CN_BTYPE

CREATE_CONTAINER. Allow a user to create a new container for another user

CREATE_CONTAINER_ITEM. Allow a user to create a container item for another user

CREATE_COPY. Allow a user to create a new copy object

CREATE_COPY_BTYPE. CREATE_COPY_BTYPE

CREATE_COPY_LOCATION. Allow a user to create a new copy location

CREATE_COPY_NOTE. Allow a user to create a new copy note

CREATE_COPY_STAT_CAT. User may create a copy statistical category

CREATE_COPY_STAT_CAT_ENTRY. User may create an entry in a copy statistical category

CREATE_COPY_STAT_CAT_ENTRY_MAP. User may link a copy to an entry in a statistical category

CREATE_COPY_STATUS. CREATE_COPY_STATUS

CREATE_COPY_TRANSIT. Allow a user to create a transit_copy object for transiting a copy

CREATE_DUPLICATE_HOLDS. Allow a user to create duplicate holds (two or more holds on the same title)

CREATE_FUND. Allow a user to create a new fund

CREATE_FUND_ALLOCATION. Allow a user to create a new fund allocation

CREATE_FUNDING_SOURCE. Allow a user to create a new funding source

CREATE_HOLD_NOTIFICATION. Allow a user to create new hold notifications

CREATE_HOURS_OF_OPERATION. CREATE_HOURS_OF_OPERATION

CREATE_IMPORT_ITEM. CREATE_IMPORT_ITEM

CREATE_IMPORT_ITEM_ATTR_DEF. CREATE_IMPORT_ITEM_ATTR_DEF

CREATE_IMPORT_TRASH_FIELD. CREATE_IMPORT_TRASH_FIELD

CREATE_IN_HOUSE_USE. Allow a user to create a new in-house-use

CREATE_INVOICE. CREATE_INVOICE

CREATE_INVOICE_ITEM_TYPE. CREATE_INVOICE_ITEM_TYPE

CREATE_INVOICE_METHOD. CREATE_INVOICE_METHOD

CREATE_ITEM_FORM. CREATE_ITEM_FORM

CREATE_ITEM_TYPE. CREATE_ITEM_TYPE

CREATE_LANGUAGE. CREATE_LANGUAGE

CREATE_LASSO. CREATE_LASSO

CREATE_LASSO_MAP. CREATE_LASSO_MAP

CREATE_LIT_FORM. CREATE_LIT_FORM

CREATE_LOCALE. CREATE_LOCALE

CREATE_MARC. Allow a user to create new MARC records

CREATE_MARC_CODE. CREATE_MARC_CODE

CREATE_MERGE_PROFILE. CREATE_MERGE_PROFILE

CREATE_METABIB_CLASS. CREATE_METABIB_CLASS

CREATE_METABIB_FIELD. CREATE_METABIB_FIELD

CREATE_METABIB_SEARCH_ALIAS. CREATE_METABIB_SEARCH_ALIAS

CREATE_MFHD_RECORD. Allows a user to create a new MFHD record

CREATE_MY_CONTAINER. Allow a user to create a container for themselves

CREATE_NET_ACCESS_LEVEL. CREATE_NET_ACCESS_LEVEL

CREATE_NON_CAT_TYPE. Allow a user to create a new non-cataloged item type

CREATE_ORG_ADDRESS. CREATE_ORG_ADDRESS

CREATE_ORG_TYPE. CREATE_ORG_TYPE

CREATE_ORG_UNIT. CREATE_ORG_UNIT

CREATE_ORG_UNIT_CLOSING. CREATE_ORG_UNIT_CLOSING

CREATE_PATRON_STAT_CAT. User may create a new patron statistical category

CREATE_PATRON_STAT_CAT_ENTRY. User may create an entry in a patron statistical category

CREATE_PATRON_STAT_CAT_ENTRY_MAP. User may link another user to an entry in a statistical category

CREATE_PAYMENT. Allow a user to record payments in the Billing Interface

CREATE_PERM. CREATE_PERM

CREATE_PICKLIST. Allows a user to create a picklist

CREATE_PROVIDER. Allow a user to create a new provider

CREATE_PURCHASE_ORDER. Allows a user to create a purchase order

CREATE_RELEVANCE_ADJUSTMENT. CREATE_RELEVANCE_ADJUSTMENT

CREATE_SURVEY. CREATE_SURVEY

CREATE_TITLE_NOTE. Allow a user to create a new title note

CREATE_TRANSACTION. Allow a user to create a new billable transaction

CREATE_TRANSIT. Allow a user to place an item in transit

CREATE_TRANSLATION. CREATE_TRANSLATION

CREATE_TRIGGER_CLEANUP. Allow a user to create trigger cleanup entries

CREATE_TRIGGER_EVENT_DEF. Allow a user to create trigger event definitions

CREATE_TRIGGER_HOOK. Allow a user to create trigger hooks

CREATE_TRIGGER_REACTOR. Allow a user to create trigger reactors

CREATE_TRIGGER_VALIDATOR. Allow a user to create trigger validators

CREATE_USER. Allow a user to create another user

CREATE_USER_BTYPE. CREATE_USER_BTYPE

CREATE_USER_GROUP_LINK. Allow a user to add other users to permission groups

CREATE_VOLUME. Allow a user to create a volume

CREATE_VOLUME_NOTE. Allow a user to create a new volume note

CREATE_VR_FORMAT. CREATE_VR_FORMAT

CREATE_XML_TRANSFORM. CREATE_XML_TRANSFORM

DELETE_ACQ_FUNDING_SOURCE. DELETE_ACQ_FUNDING_SOURCE

DELETE_AUDIENCE. DELETE_AUDIENCE

DELETE_AUTHORITY_IMPORT_IMPORT_FIELD_DEF.
DELETE_AUTHORITY_IMPORT_IMPORT_FIELD_DEF

DELETE_AUTHORITY_IMPORT_QUEUE. DELETE_AUTHORITY_IMPORT_QUEUE

DELETE_AUTHORITY_RECORD_NOTE. DELETE_AUTHORITY_RECORD_NOTE

DELETE_BIB_BTYPE. DELETE_BIB_BTYPE

DELETE_BIB_IMPORT_IMPORT_FIELD_DEF. DELETE_BIB_IMPORT_IMPORT_FIELD_DEF

DELETE_BIB_IMPORT_QUEUE. DELETE_BIB_IMPORT_QUEUE

DELETE_BIB_LEVEL. DELETE_BIB_LEVEL

DELETE_BIBLIO_FINGERPRINT. DELETE_BIBLIO_FINGERPRINT

DELETE_BIB_SOURCE. DELETE_BIB_SOURCE

DELETE_BILLING_TYPE. DELETE_BILLING_TYPE

DELETE_CIRC_DURATION. DELETE_CIRC_DURATION

DELETE_CIRC_MOD. DELETE_CIRC_MOD

DELETE_CN_BTYPE. DELETE_CN_BTYPE

DELETE_CONTAINER. Allow a user to delete another user's container

DELETE_CONTAINER_ITEM. Allow a user to delete an item out of another user's container

DELETE_COPY. Allow a user to delete a copy

DELETE_COPY_BTYPE. DELETE_COPY_BTYPE

DELETE_COPY_LOCATION. Allow a user to delete a copy location

DELETE_COPY_NOTE. Allow a user to delete another user's copy notes

DELETE_COPY_STAT_CAT. User may delete a copy statistical category

DELETE_COPY_STAT_CAT_ENTRY. User may delete an entry from a copy statistical category

DELETE_COPY_STAT_CAT_ENTRY_MAP. User may delete a copy statistical category entry map

DELETE_COPY_STATUS. DELETE_COPY_STATUS

DELETE_FUND. Allow a user to delete a fund

DELETE_FUND_ALLOCATION. Allow a user to delete a fund allocation

DELETE_FUNDING_SOURCE. Allow a user to delete a funding source

DELETE_HOLDS. * no longer applicable

DELETE_HOURS_OF_OPERATION. DELETE_HOURS_OF_OPERATION

DELETE_IMPORT_ITEM. DELETE_IMPORT_ITEM

DELETE_IMPORT_ITEM_ATTR_DEF. DELETE_IMPORT_ITEM_ATTR_DEF

DELETE_IMPORT_TRASH_FIELD. DELETE_IMPORT_TRASH_FIELD

DELETE_INVOICE_ITEM_TYPE. DELETE_INVOICE_ITEM_TYPE

DELETE_INVOICE_METHOD. DELETE_INVOICE_METHOD

DELETE_ITEM_FORM. DELETE_ITEM_FORM

DELETE_ITEM_TYPE. DELETE_ITEM_TYPE

DELETE_LANGUAGE. DELETE_LANGUAGE

DELETE_LASSO. DELETE_LASSO

DELETE_LASSO_MAP. DELETE_LASSO_MAP

DELETE_LIT_FORM. DELETE_LIT_FORM

DELETE_LOCALE. DELETE_LOCALE

DELETE_MARC_CODE. DELETE_MARC_CODE

DELETE_MERGE_PROFILE. DELETE_MERGE_PROFILE

DELETE_METABIB_CLASS. DELETE_METABIB_CLASS

DELETE_METABIB_FIELD. DELETE_METABIB_FIELD

DELETE_METABIB_SEARCH_ALIAS. DELETE_METABIB_SEARCH_ALIAS

DELETE_MFHD_RECORD. Allows a user to delete an MFHD record

DELETE_NET_ACCESS_LEVEL. DELETE_NET_ACCESS_LEVEL

DELETE_NON_CAT_TYPE. Allow a user to delete a non cataloged type

DELETE_ORG_ADDRESS. DELETE_ORG_ADDRESS

DELETE_ORG_TYPE. DELETE_ORG_TYPE

DELETE_ORG_UNIT. DELETE_ORG_UNIT

DELETE_ORG_UNIT_CLOSING. DELETE_ORG_UNIT_CLOSING

DELETE_PATRON_STAT_CAT. User may delete a patron statistical category

DELETE_PATRON_STAT_CAT_ENTRY. User may delete an entry from a patron statistical category

DELETE_PATRON_STAT_CAT_ENTRY_MAP. User may delete a patron statistical category entry map

DELETE_PERM. DELETE_PERM

DELETE_PROVIDER. Allow a user to delate a provider

DELETE_RECORD. Allow a staff member to directly remove a bibliographic record

DELETE_RELEVANCE_ADJUSTMENT. DELETE_RELEVANCE_ADJUSTMENT

DELETE_SURVEY. DELETE_SURVEY

DELETE_TITLE_NOTE. Allow a user to delete another user's title note

DELETE_TRANSIT. DELETE_TRANSIT

DELETE_TRANSLATION. DELETE_TRANSLATION

DELETE_TRIGGER_CLEANUP. Allow a user to delete trigger cleanup entries

DELETE_TRIGGER_EVENT_DEF. Allow a user to delete trigger event definitions

DELETE_TRIGGER_HOOK. Allow a user to delete trigger hooks

DELETE_TRIGGER_REACTOR. Allow a user to delete trigger reactors

DELETE_TRIGGER_TEMPLATE_OUTPUT. Allow a user to delete trigger template output

DELETE_TRIGGER_VALIDATOR. Allow a user to delete trigger validators

DELETE_USER. Allow a user to mark a user as deleted

DELETE_USER_BTYPE. DELETE_USER_BTYPE

DELETE_VOLUME. Allow a user to delete a volume

DELETE_VOLUME_NOTE. Allow a user to delete another user's volume note

DELETE_VR_FORMAT. DELETE_VR_FORMAT

DELETE_WORKSTATION. Allow a user to remove an existing workstation so a new one can replace it

DELETE_XML_TRANSFORM. DELETE_XML_TRANSFORM

GENERAL_ACQ. Lowest level permission required to access the ACQ interface

group_application.user. Allow a user to add/remove users to/from the "User" group

group_application.user.patron. Allow a user to add/remove users to/from the "Patron" group

group_application.user.sip_client. Allow a user to add/remove users to/from the "SIP-Client" group

group_application.user.staff. Allow a user to add/remove users to/from the "Staff" group

group_application.user.staff.acq. Allows a user to add/remove/edit users in the "ACQ" group

group_application.user.staff.acq_admin. Allows a user to add/remove/edit users in the "Acquisitions Administrator" group

group_application.user.staff.admin.global_admin. Allow a user to add/remove users to/from the "GlobalAdmin" group

group_application.user.staff.admin.lib_manager. Allow a user to add/remove users to/from the "LibraryManager" group

group_application.user.staff.admin.local_admin. Allow a user to add/remove users to/from the "LocalAdmin" group

group_application.user.staff.cat. Allow a user to add/remove users to/from the "Cataloger" group

group_application.user.staff.cat.cat1. Allow a user to add/remove users to/from the "Cat1" group

group_application.user.staff.circ. Allow a user to add/remove users to/from the "Circulator" group

group_application.user.staff.supercat. Allow a user to add/remove users to/from the "Supercat" group

group_application.user.vendor. Allow a user to add/remove users to/from the "Vendor" group

HOLD_EXISTS.override. Allow a user to place multiple holds on a single title

HOLD_ITEM_CHECKED_OUT.override. Allows a user to place a hold on an item that they already have checked out

HOLD_LOCAL_AVAIL_OVERRIDE. Allow a user to place a hold despite the availability of a local copy

IMPORT_ACQ_LINEITEM_BIB_RECORD. Allows a user to import a bib record from the acq staging area (on-order record) into the ILS bib data set

IMPORT_MARC. Allow a user to import a MARC record via the Z39.50 interface

ISSUANCE_HOLDS. Allow a user to place holds on serials issuances

ITEM_AGE_PROTECTED.override. Allow a user to place a hold on an age-protected item

ITEM_ON_HOLDS_SHELF.override. Allow staff to override item on holds shelf failure

MANAGE_CLAIM. MANAGE_CLAIM

MANAGE_FUND. Allow a user to view/credit/debit a fund

MANAGE_FUNDING_SOURCE. Allow a user to view/credit/debit a funding source

MANAGE_PROVIDER. Allow a user to view and purchase from a provider

MARK_BAD_DEBT. Allow a user to mark a transaction as bad (unrecoverable) debt

MARK_ITEM_AVAILABLE. Allow a user to mark an item status as 'available'

MARK_ITEM_BINDERY. Allow a user to mark an item status as 'bindery'

MARK_ITEM_CHECKED_OUT. Allow a user to mark an item status as 'checked out'

MARK_ITEM_ILL. Allow a user to mark an item status as 'inter-library loan'

MARK_ITEM_IN_PROCESS. Allow a user to mark an item status as 'in process'

MARK_ITEM_IN_TRANSIT. Allow a user to mark an item status as 'in transit'

MARK_ITEM_LOST. Allow a user to mark an item status as 'lost'

MARK_ITEM_MISSING. Allow a user to mark an item status as 'missing'

MARK_ITEM_ON_HOLDS_SHELF. Allow a user to mark an item status as 'on holds shelf'

MARK_ITEM_ON_ORDER. Allow a user to mark an item status as 'on order'

MARK_ITEM_RESHELVING. Allow a user to mark an item status as 'reshelving'

MAX_RENEWALS_REACHED.override. Allow a user to renew an item past the maximum renewal count

MERGE_AUTH_RECORDS. Allow a user to merge authority records together

MERGE_BIB_RECORDS. MERGE_BIB_RECORDS

MERGE_USERS. MERGE_USERS

money.collections_tracker.create. Allow a user to put someone into collections

money.collections_tracker.delete. Allow a user to remove someone from collections

MR_HOLDS. Allow a user to create a metarecord holds

OFFLINE_EXECUTE. Allow a user to execute an offline script batch

OFFLINE_UPLOAD. Allow a user to upload an offline script

OFFLINE_VIEW. Allow a user to view uploaded offline script information

OPAC_LOGIN. Allow a user to log in to the OPAC

OVERRIDE_HOLD_HAS_LOCAL_COPY. Allow a user to override the circ.holds.hold_has_copy_at.block setting

PATRON_EXCEEDS_CHECKOUT_COUNT.override. Allow staff to override checkout count failure

PATRON_EXCEEDS_FINES.override. Allow staff to override fine amount checkout failure

PATRON_EXCEEDS_OVERDUE_COUNT.override. Allow staff to override overdue count failure

RECEIVE_PURCHASE_ORDER. Allows a user to mark a purchase order, lineitem, or individual copy as received

RECEIVE_SERIAL. Receive serial items

REGISTER_WORKSTATION. Allow a user to register a new workstation

REMOTE_Z3950_QUERY. Allow a user to perform Z39.50 queries against remote servers

REMOVE_GROUP_PERM. REMOVE_GROUP_PERM

REMOVE_USER_GROUP_LINK. Allow a user to remove other users from permission groups

RENEW_CIRC. Allow a user to renew items

RENEW_HOLD_OVERRIDE. Allow a user to continue to renew an item even if it is required for a hold

REQUEST_HOLDS. Allow a user to create holds for another user (if true, we still check to make sure they have permission to make the type of hold they are requesting, for example, COPY_HOLDS)

REQUEST_HOLDS_OVERRIDE. * no longer applicable

RETRIEVE_RESERVATION_PULL_LIST. Allows a user to retrieve a booking reservation pull list

RUN_REPORTS. Allow a user to run reports

SET_CIRC_CLAIMS_RETURNED. Allow a user to mark an item as 'claims returned'

SET_CIRC_CLAIMS_RETURNED.override. Allows staff to override the max claims returned value for a patron

SET_CIRC_LOST. Allow a user to mark an item as 'lost'

SET_CIRC_MISSING. Allow a user to mark an item as 'missing'

SHARE_REPORT_FOLDER. Allow a user to share report his own folders

STAFF_LOGIN. Allow a user to log in to the staff client

TITLE_HOLDS. Allow a user to place a hold at the title level

TRANSIT_COPY. TRANSIT_COPY

UNBAR_PATRON. Allow a user to un-bar a patron

UPDATE_ACQ_FUNDING_SOURCE. UPDATE_ACQ_FUNDING_SOURCE

UPDATE_AUDIENCE. UPDATE_AUDIENCE

UPDATE_AUTHORITY_IMPORT_IMPORT_FIELD_DEF. UPDATE_AUTHORITY_IMPORT_IMPORT_FIELD_DEF

UPDATE_AUTHORITY_IMPORT_QUEUE. UPDATE_AUTHORITY_IMPORT_QUEUE

UPDATE_AUTHORITY_RECORD_NOTE. UPDATE_AUTHORITY_RECORD_NOTE

UPDATE_BATCH_COPY. Allow a user to edit copies in batch

UPDATE_BIB_BTYPE. UPDATE_BIB_BTYPE

UPDATE_BIB_IMPORT_IMPORT_FIELD_DEF. UPDATE_BIB_IMPORT_IMPORT_FIELD_DEF

UPDATE_BIB_IMPORT_QUEUE. UPDATE_BIB_IMPORT_QUEUE

UPDATE_BIB_LEVEL. UPDATE_BIB_LEVEL

UPDATE_BIBLIO_FINGERPRINT. UPDATE_BIBLIO_FINGERPRINT

UPDATE_BIB_SOURCE. UPDATE_BIB_SOURCE

UPDATE_BILLING_TYPE. UPDATE_BILLING_TYPE

UPDATE_BILL_NOTE. Allows staff to edit the note for a bill on a transaction

UPDATE_CIRC_DURATION. UPDATE_CIRC_DURATION

UPDATE_CIRC_MOD. UPDATE_CIRC_MOD

UPDATE_CN_BTYPE. UPDATE_CN_BTYPE

UPDATE_CONTAINER. Allow a user to update another user's container

UPDATE_COPY. Allow a user to edit a copy

UPDATE_COPY_BTYPE. UPDATE_COPY_BTYPE

UPDATE_COPY_LOCATION. Allow a user to update a copy location

UPDATE_COPY_NOTE. UPDATE_COPY_NOTE

UPDATE_COPY_STAT_CAT. User may update a copy statistical category

UPDATE_COPY_STAT_CAT_ENTRY. User may update an entry in a copy statistical category

UPDATE_COPY_STATUS. UPDATE_COPY_STATUS

UPDATE_FUND. Allow a user to update a fund

UPDATE_FUND_ALLOCATION. Allow a user to update a fund allocation

UPDATE_FUNDING_SOURCE. Allow a user to update a funding source

UPDATE_GROUP_PERM. UPDATE_GROUP_PERM

UPDATE_HOLD. Allow a user to update another user's hold

UPDATE_HOURS_OF_OPERATION. UPDATE_HOURS_OF_OPERATION

UPDATE_IMPORT_ITEM. UPDATE_IMPORT_ITEM

UPDATE_IMPORT_ITEM_ATTR_DEF. UPDATE_IMPORT_ITEM_ATTR_DEF

UPDATE_IMPORT_TRASH_FIELD. UPDATE_IMPORT_TRASH_FIELD

UPDATE_INVOICE_ITEM_TYPE. UPDATE_INVOICE_ITEM_TYPE

UPDATE_INVOICE_METHOD. UPDATE_INVOICE_METHOD

UPDATE_ITEM_FORM. UPDATE_ITEM_FORM

UPDATE_ITEM_TYPE. UPDATE_ITEM_TYPE

UPDATE_LANGUAGE. UPDATE_LANGUAGE

UPDATE_LASSO. UPDATE_LASSO

UPDATE_LASSO_MAP. UPDATE_LASSO_MAP

UPDATE_LIT_FORM. UPDATE_LIT_FORM

UPDATE_LOCALE. UPDATE_LOCALE

UPDATE_MARC. Allow a user to edit a MARC record

UPDATE_MARC_CODE. UPDATE_MARC_CODE

UPDATE_MERGE_PROFILE. UPDATE_MERGE_PROFILE

UPDATE_METABIB_CLASS. UPDATE_METABIB_CLASS

UPDATE_METABIB_FIELD. UPDATE_METABIB_FIELD

UPDATE_METABIB_SEARCH_ALIAS. UPDATE_METABIB_SEARCH_ALIAS

UPDATE_MFHD_RECORD. Allows a user to update an MFHD record

UPDATE_NET_ACCESS_LEVEL. UPDATE_NET_ACCESS_LEVEL

UPDATE_NON_CAT_TYPE. Allow a user to update a non-cataloged item type

UPDATE_ORG_ADDRESS. UPDATE_ORG_ADDRESS

UPDATE_ORG_SETTING. Allow a user to update an organization unit setting

UPDATE_ORG_TYPE. UPDATE_ORG_TYPE

UPDATE_ORG_UNIT. Allow a user to change the settings for an organization unit

UPDATE_ORG_UNIT_CLOSING. UPDATE_ORG_UNIT_CLOSING

UPDATE_ORG_UNIT_SETTING_ALL. UPDATE_ORG_UNIT_SETTING_ALL

UPDATE_ORG_UNIT_SETTING.auth.opac_timeout.
UPDATE_ORG_UNIT_SETTING.auth.opac_timeout

UPDATE_ORG_UNIT_SETTING.auth.staff_timeout.
UPDATE_ORG_UNIT_SETTING.auth.staff_timeout

UPDATE_ORG_UNIT_SETTING.cat.bib.alert_on_empty.
UPDATE_ORG_UNIT_SETTING.cat.bib.alert_on_empty

UPDATE_ORG_UNIT_SETTING.cat.bib.keep_on_empty.
UPDATE_ORG_UNIT_SETTING.cat.bib.keep_on_empty

UPDATE_ORG_UNIT_SETTING.cat.default_item_price.
UPDATE_ORG_UNIT_SETTING.cat.default_item_price

UPDATE_ORG_UNIT_SETTING.circ.block_renews_for_holds. Allow a user to enable blocking of renews on items that could fulfill holds

UPDATE_ORG_UNIT_SETTING.circ.hold_boundary.hard.
UPDATE_ORG_UNIT_SETTING.circ.hold_boundary.hard

UPDATE_ORG_UNIT_SETTING.circ.hold_boundary.soft.
UPDATE_ORG_UNIT_SETTING.circ.hold_boundary.soft

UPDATE_ORG_UNIT_SETTING.circ.hold_expire_alert_interval.
UPDATE_ORG_UNIT_SETTING.circ.hold_expire_alert_interval

UPDATE_ORG_UNIT_SETTING.circ.hold_expire_interval.
UPDATE_ORG_UNIT_SETTING.circ.hold_expire_interval

UPDATE_ORG_UNIT_SETTING.circ.hold_stalling.soft.
UPDATE_ORG_UNIT_SETTING.circ.hold_stalling.soft

UPDATE_ORG_UNIT_SETTING.circ.item_checkout_history.max.
UPDATE_ORG_UNIT_SETTING.circ.item_checkout_history.max

UPDATE_ORG_UNIT_SETTING.circ.lost_materials_processing_fee.
UPDATE_ORG_UNIT_SETTING.circ.lost_materials_processing_fee

UPDATE_ORG_UNIT_SETTING.circ.reshelving_complete.interval.
UPDATE_ORG_UNIT_SETTING.circ.reshelving_complete.interval

UPDATE_ORG_UNIT_SETTING.circ.selfcheck.alert_on_checkout_event.
UPDATE_ORG_UNIT_SETTING.circ.selfcheck.alert_on_checkout_event

UPDATE_ORG_UNIT_SETTING.circ.selfcheck.patron_login_timeout.
UPDATE_ORG_UNIT_SETTING.circ.selfcheck.patron_login_timeout

UPDATE_ORG_UNIT_SETTING.circ.selfcheck.require_patron_password.
UPDATE_ORG_UNIT_SETTING.circ.selfcheck.require_patron_password

UPDATE_ORG_UNIT_SETTING.circ.void_overdue_on_lost.
UPDATE_ORG_UNIT_SETTING.circ.void_overdue_on_lost

UPDATE_ORG_UNIT_SETTING.credit.payments.allow.
UPDATE_ORG_UNIT_SETTING.credit.payments.allow

UPDATE_ORG_UNIT_SETTING.global.juvenile_age_threshold.
UPDATE_ORG_UNIT_SETTING.global.juvenile_age_threshold

UPDATE_ORG_UNIT_SETTING.global.password_regex.
UPDATE_ORG_UNIT_SETTING.global.password_regex

UPDATE_ORG_UNIT_SETTING.opac.barcode_regex.
UPDATE_ORG_UNIT_SETTING.opac.barcode_regex

UPDATE_ORG_UNIT_SETTING.org.bounced_emails.
UPDATE_ORG_UNIT_SETTING.org.bounced_emails

UPDATE_ORG_UNIT_SETTING.patron.password.use_phone.
UPDATE_ORG_UNIT_SETTING.patron.password.use_phone

UPDATE_PATRON_CLAIM_NEVER_CHECKED_OUT_COUNT. Allows staff to manually change a patron's claims never checkout out count

UPDATE_PATRON_CLAIM_RETURN_COUNT. Allows staff to manually change a patron's claims returned count

UPDATE_PATRON_STAT_CAT. User may update a patron statistical category

UPDATE_PATRON_STAT_CAT_ENTRY. User may update an entry in a patron statistical category

UPDATE_PAYMENT_NOTE. Allows staff to edit the note for a payment on a transaction

UPDATE_PERM. UPDATE_PERM

UPDATE_PICKUP_LIB_FROM_HOLDS_SHELF. UPDATE_PICKUP_LIB_FROM_HOLDS_SHELF

UPDATE_PICKUP_LIB_FROM_TRANSIT. Allow a user to change the pickup and transit destination for a captured hold item already in transit

UPDATE_PROVIDER. Allow a user to update a provider

UPDATE_RECORD. Allow a user to update and undelete bibliographic records

UPDATE_RELEVANCE_ADJUSTMENT. UPDATE_RELEVANCE_ADJUSTMENT

UPDATE_SURVEY. UPDATE_SURVEY

UPDATE_TRANSIT. UPDATE_TRANSIT

UPDATE_TRANSLATION. UPDATE_TRANSLATION

UPDATE_TRIGGER_CLEANUP. Allow a user to update trigger cleanup entries

UPDATE_TRIGGER_EVENT_DEF. Allow a user to update trigger event definitions

UPDATE_TRIGGER_HOOK. Allow a user to update trigger hooks

UPDATE_TRIGGER_REACTOR. Allow a user to update trigger reactors

UPDATE_TRIGGER_VALIDATOR. Allow a user to update trigger validators

UPDATE_USER. Allow a user to edit a user's record

UPDATE_USER_BTYPE. UPDATE_USER_BTYPE

UPDATE_VOLUME. Allow a user to edit volumes - needed for merging records. This is a duplicate of VOLUME_UPDATE; user must have both permissions at appropriate level to merge records.

UPDATE_VOLUME_NOTE. UPDATE_VOLUME_NOTE

UPDATE_VR_FORMAT. UPDATE_VR_FORMAT

UPDATE_XML_TRANSFORM. UPDATE_XML_TRANSFORM

user_request.create. user_request.create

user_request.delete. user_request.delete

user_request.update. user_request.update

user_request.view. user_request.view

VIEW_ACQ_FUND_ALLOCATION_PERCENT. VIEW_ACQ_FUND_ALLOCATION_PERCENT

VIEW_ACQ_FUNDING_SOURCE. VIEW_ACQ_FUNDING_SOURCE

VIEW_AUTHORITY_RECORD_NOTES. VIEW_AUTHORITY_RECORD_NOTES

VIEW_BILLING_TYPE. Allow a user to view billing types

VIEW_CIRC_MATRIX_MATCHPOINT. VIEW_CIRC_MATRIX_MATCHPOINT

VIEW_CIRCULATIONS. Allow a user to see what another user has checked out

VIEW_CLAIM. VIEW_CLAIM

VIEW_CONTAINER. Allow a user to view another user's containers (buckets)

VIEW_COPY_CHECKOUT_HISTORY. Allow a user to view which users have checked out a given copy

VIEW_COPY_NOTES. Allow a user to view all notes attached to a copy

VIEW_CREDIT_CARD_PROCESSING. View org unit settings related to credit card processing

VIEW_FUND. Allow a user to view a fund

VIEW_FUND_ALLOCATION. Allow a user to view a fund allocation

VIEW_FUNDING_SOURCE. Allow a user to view a funding source

VIEW_GROUP_PENALTY_THRESHOLD. VIEW_GROUP_PENALTY_THRESHOLD

VIEW_HOLD. Allow a user to view another user's holds

VIEW_HOLD_MATRIX_MATCHPOINT. VIEW_HOLD_MATRIX_MATCHPOINT

VIEW_HOLD_NOTIFICATION. Allow a user to view notifications attached to a hold

VIEW_HOLD_PERMIT. Allow a user to see if another user has permission to place a hold on a given copy

VIEW_INVOICE. VIEW_INVOICE

VIEW_MERGE_PROFILE. VIEW_MERGE_PROFILE

VIEW_ORG_SETTINGS. Allows a user to view all org settings at the specified level

VIEW_PERM_GROUPS. Allow a user to view other users' permission groups

VIEW_PERMISSION. Allow a user to view user permissions within the user permissions editor

VIEW_PERMIT_CHECKOUT. Allow a user to determine whether another user can check out an item

VIEW_PICKLIST. Allow a user to view another users picklist

VIEW_PROVIDER. Allow a user to view a provider

VIEW_PURCHASE_ORDER. Allows a user to view a purchase order

VIEW_REPORT_OUTPUT. Allow a user to view report output

VIEW_SERIAL_SUBSCRIPTION. VIEW_SERIAL_SUBSCRIPTION

VIEW_STANDING_PENALTY. VIEW_STANDING_PENALTY

VIEW_TITLE_NOTES. Allow a user to view all notes attached to a title

VIEW_TRANSACTION. Allow a user may view another user's transactions

VIEW_TRIGGER_EVENT_DEF. Allow a user to view trigger event definitions

VIEW_USER. Allow a user to view another user's Patron Record

VIEW_USER_FINES_SUMMARY. Allow a user to view bill details

VIEW_USER_TRANSACTIONS. Allow a user to see another user's grocery or circulation transactions in the Bills Interface; duplicate of VIEW_TRANSACTION

VIEW_VOLUME_NOTES. Allow a user to view all notes attached to a volume

VIEW_ZIP_DATA. Allow a user to query the ZIP code data method

VOID_BILLING. Allow a user to void a bill

VOLUME_HOLDS. Allow a user to place a volume level hold

Chapter 26. Database Schema

This is the schema for the Evergreen database.

Schema acq

Table: acq_lineitem_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- PRIMARY KEY,
audit_time : timestamp with time zone -- NOT NULL,
audit_action : text -- NOT NULL,
id : bigint -- NOT NULL,
creator : integer -- NOT NULL,
editor : integer -- NOT NULL,
selector : integer -- NOT NULL,
provider : integer --
purchase_order : integer --
picklist : integer --
expected_recv_time : timestamp with time zone --
create_time : timestamp with time zone -- NOT NULL,
edit_time : timestamp with time zone -- NOT NULL,
marc : text -- NOT NULL,
eg_bib_id : bigint --
source_label : text --
state : text -- NOT NULL,
cancel_reason : integer --
estimated_unit_price : numeric --
claim_policy : integer --

Indexes:

acq_lineitem_hist_id_idx : id

View: acq_lineitem_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : bigint --
creator : integer --
editor : integer --

selector : integer --
provider : integer --
purchase_order : integer --
picklist : integer --
expected_recv_time : timestamp with time zone --
create_time : timestamp with time zone --
edit_time : timestamp with time zone --
marc : text --
eg_bib_id : bigint --
source_label : text --
state : text --
cancel_reason : integer --
estimated_unit_price : numeric --
claim_policy : integer --

Table: acq_purchase_order_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- PRIMARY KEY,
audit_time : timestamp with time zone -- NOT NULL,
audit_action : text -- NOT NULL,
id : integer -- NOT NULL,
owner : integer -- NOT NULL,
creator : integer -- NOT NULL,
editor : integer -- NOT NULL,
ordering_agency : integer -- NOT NULL,
create_time : timestamp with time zone -- NOT NULL,
edit_time : timestamp with time zone -- NOT NULL,
provider : integer -- NOT NULL,
state : text -- NOT NULL,
order_date : timestamp with time zone --
name : text -- NOT NULL,
cancel_reason : integer --
prepayment_required : boolean -- NOT NULL,

Indexes:

acq_po_hist_id_idx : id

View: acq_purchase_order_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : integer --

owner : integer --
creator : integer --
editor : integer --
ordering_agency : integer --
create_time : timestamp with time zone --
edit_time : timestamp with time zone --
provider : integer --
state : text --
order_date : timestamp with time zone --
name : text --
cancel_reason : integer --
prepayment_required : boolean --

View: all_fund_allocation_total

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric --

View: all_fund_combined_balance

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric --

View: all_fund_encumbrance_total

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric --

View: all_fund_spent_balance

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric --

View: all_fund_spent_total

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric --

Table: cancel_reason

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
org_unit : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.org_unit.
label : text -- `UNIQUE#1`, `NOT NULL`,
description : text -- `NOT NULL`,
keep_debits : boolean -- `NOT NULL`, `DEFAULT false`,

Tables referencing acq.lineitem via Foreign Key Constraints:

acq.lineitem
acq.lineitem_detail
acq.purchase_order
acq.user_request

Table: claim

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
type : integer -- `NOT NULL`, `REFERENCES` acq.claim_type.
lineitem_detail : bigint -- `NOT NULL`, `REFERENCES` acq.lineitem_detail.

Indexes:

claim_lid_idx : lineitem_detail

Tables referencing acq.claim_event via Foreign Key Constraints:

acq.claim_event

Table: claim_event

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
type : integer -- `NOT NULL`, `REFERENCES` acq.claim_event_type.
claim : serial -- `NOT NULL`, `REFERENCES` acq.claim.
event_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
creator : integer -- `NOT NULL`, `REFERENCES` actor.usr.
note : text --

Indexes:

claim_event_claim_date_idx : claim, event_date

Table: claim_event_type

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
org_unit : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
code : text -- UNIQUE#1, NOT NULL,
description : text -- NOT NULL,
library_initiated : boolean -- NOT NULL, DEFAULT false,

Tables referencing acq.claim_event via Foreign Key Constraints:

acq.claim_event
acq.claim_policy_action
acq.serial_claim_event

Table: claim_policy

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
org_unit : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
name : text -- UNIQUE#1, NOT NULL,
description : text -- NOT NULL,

Tables referencing acq.claim_policy_action via Foreign Key Constraints:

acq.claim_policy_action
acq.lineitem
acq.provider

Table: claim_policy_action

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
claim_policy : integer -- UNIQUE#1, NOT NULL, REFERENCES acq.claim_policy.
action_interval : interval -- UNIQUE#1, NOT NULL,
action : integer -- NOT NULL, REFERENCES acq.claim_event_type.

Table: claim_type

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
org_unit : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.

code : text -- UNIQUE#1, NOT NULL,
description : text -- NOT NULL,

Tables referencing acq.claim via Foreign Key Constraints:

acq.claim
acq.serial_claim

Table: currency_type

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
label : text --

Tables referencing acq.exchange_rate via Foreign Key Constraints:

acq.exchange_rate
acq.fund
acq.fund_debit
acq.funding_source
acq.provider

Table: debit_attribution

Columns:

field name : datatype -- parameters, contraints and notes
id : integer -- PRIMARY KEY,
fund_debit : integer -- NOT NULL, REFERENCES acq.fund_debit.
debit_amount : numeric -- NOT NULL,
funding_source_credit : integer -- REFERENCES acq.funding_source_credit.
credit_amount : numeric --

Indexes:

acq_attribution_credit_idx : funding_source_credit
acq_attribution_debit_idx : fund_debit

Table: distribution_formula

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
name : text -- UNIQUE#1, NOT NULL,
skip_count : integer -- NOT NULL,

Tables referencing acq.distribution_formula_application via Foreign Key Constraints:

acq.distribution_formula_application
acq.distribution_formula_entry

Table: distribution_formula_application

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
creator : integer -- NOT NULL, REFERENCES actor.usr.
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
formula : integer -- NOT NULL, REFERENCES acq.distribution_formula.
lineitem : integer -- NOT NULL, REFERENCES acq.lineitem.

Indexes:

acqdfa_creator_idx : creator
acqdfa_df_idx : formula
acqdfa_li_idx : lineitem

Table: distribution_formula_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
formula : integer -- UNIQUE#1, NOT NULL, REFERENCES acq.distribution_formula.
position : integer -- UNIQUE#1, NOT NULL,
item_count : integer -- NOT NULL,
owning_lib : integer -- REFERENCES actor.org_unit.
location : integer -- REFERENCES asset.copy_location.

Constraints:

acqdfe_must_be_somewhere : CHECK (((owning_lib IS NOT NULL) OR (location IS NOT NULL)))

Table: edi_account

Columns:

field name : datatype -- parameters, contraints and notes
id : integer -- PRIMARY KEY, DEFAULT nextval('config.remote_account_id_seq'::regclass),
label : text -- NOT NULL,
host : text -- NOT NULL,
username : text --
password : text --
account : text --
path : text --
owner : integer -- NOT NULL,
last_activity : timestamp with time zone --
provider : integer -- NOT NULL, REFERENCES acq.provider.
in_dir : text --
vendcode : text --

vendacct : text --

Tables referencing acq.edi_message via Foreign Key Constraints:

acq.edi_message acq.provider

Table: edi_message

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
account : integer -- REFERENCES acq.edi_account.
remote_file : text --
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
translate_time : timestamp with time zone --
process_time : timestamp with time zone --
error_time : timestamp with time zone --
status : text -- NOT NULL, DEFAULT 'new'::text,
edi : text --
jedi : text --
error : text --
purchase_order : integer -- REFERENCES acq.purchase_order.
message_type : text -- NOT NULL,

Constraints:

status_value : CHECK ((status = ANY (ARRAY['new'::text, 'translated'::text, 'trans_error'::text, 'processed'::text, 'proc_error'::text, 'delete_error'::text, 'retry'::text, 'complete'::text])))
valid_message_type : CHECK ((message_type = ANY (ARRAY['ORDERS'::text, 'ORDRSP'::text, 'INVOIC'::text, 'OSTENQ'::text, 'OSTRPT'::text])))

Table: exchange_rate

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
from_currency : text -- UNIQUE#1, NOT NULL, REFERENCES acq.currency_type.
to_currency : text -- UNIQUE#1, NOT NULL, REFERENCES acq.currency_type.
ratio : numeric -- NOT NULL,

Table: fiscal_calendar

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- NOT NULL,

Tables referencing acq.fiscal_year via Foreign Key Constraints:

acq.fiscal_year actor.org_unit

Table: fiscal_year

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
calendar : integer -- UNIQUE#1, UNIQUE#2, NOT NULL, REFERENCES acq.fiscal_calendar.
year : integer -- UNIQUE#1, NOT NULL,
year_begin : timestamp with time zone -- UNIQUE#2, NOT NULL,
year_end : timestamp with time zone -- NOT NULL,

Table: fund

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
org : integer -- UNIQUE#2, UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
name : text -- UNIQUE#1, NOT NULL,
year : integer -- UNIQUE#2, UNIQUE#1, NOT NULL, DEFAULT date_part('year'::text, now()),
currency_type : text -- NOT NULL, REFERENCES acq.currency_type.
code : text -- UNIQUE#2,
rollover : boolean -- NOT NULL, DEFAULT false,
propagate : boolean -- NOT NULL, DEFAULT true,
active : boolean -- NOT NULL, DEFAULT true,
balance_warning_percent : integer --
balance_stop_percent : integer --

Constraints:

acq_fund_rollover_implies_propagate : CHECK ((propagate OR (NOT rollover)))

Tables referencing acq.fund_allocation via Foreign Key Constraints:

acq.fund_allocation
acq.fund_debit
acq.fund_tag_map
acq.fund_transfer
acq.invoice_item
acq.lineitem_detail
acq.po_item

Table: fund_allocation

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
funding_source : integer -- NOT NULL, REFERENCES acq.funding_source.
fund : integer -- NOT NULL, REFERENCES acq.fund.
amount : numeric -- NOT NULL,
allocator : integer -- NOT NULL, REFERENCES actor.usr.

note : text --
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Indexes:

fund_alloc_allocator_idx : allocator

Table: fund_allocation_percent

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
funding_source : integer -- UNIQUE#1, NOT NULL, REFERENCES acq.funding_source.
org : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
fund_code : text -- UNIQUE#1,
percent : numeric -- NOT NULL,
allocator : integer -- NOT NULL, REFERENCES actor.usr.
note : text --
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Constraints:

percentage_range : CHECK (((percent >= (0)::numeric) AND (percent <= (100)::numeric)))

View: fund_allocation_total

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric(100,2) --

View: fund_combined_balance

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric --

Table: fund_debit

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
fund : integer -- NOT NULL, REFERENCES acq.fund.
origin_amount : numeric -- NOT NULL,
origin_currency_type : text -- NOT NULL, REFERENCES acq.currency_type.

amount : numeric -- NOT NULL,
encumbrance : boolean -- NOT NULL, DEFAULT true,
debit_type : text -- NOT NULL,
xfer_destination : integer -- REFERENCES acq.fund.
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Tables referencing acq.debit_attribution via Foreign Key Constraints:

acq.debit_attribution
acq.invoice_item
acq.lineitem_detail
acq.po_item

View: fund_debit_total

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
encumbrance : boolean --
amount : numeric --

View: fund_encumbrance_total

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric --

View: fund_spent_balance

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric --

View: fund_spent_total

Columns:

field name : datatype -- parameters, contraints and notes
fund : integer --
amount : numeric --

Table: fund_tag

Columns:

field name : datatype -- parameters, contraints and notes

id : serial -- PRIMARY KEY,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
name : text -- UNIQUE#1, NOT NULL,

Tables referencing acq.fund_tag_map via Foreign Key Constraints:

acq.fund_tag_map

Table: fund_tag_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
fund : integer -- UNIQUE#1, NOT NULL, REFERENCES acq.fund.
tag : integer -- UNIQUE#1, REFERENCES acq.fund_tag.

Table: fund_transfer

Fund Transfer Each row represents the transfer of money from a source fund to a destination fund. There should be corresponding entries in acq.fund_allocation. The purpose of acq.fund_transfer is to record how much money moved from which fund to which other fund. The presence of two amount fields, rather than one, reflects the possibility that the two funds are denominated in different currencies. If they use the same currency type, the two amounts should be the same.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
src_fund : integer -- NOT NULL, REFERENCES acq.fund.
src_amount : numeric -- NOT NULL,
dest_fund : integer -- REFERENCES acq.fund.
dest_amount : numeric --
transfer_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
transfer_user : integer -- NOT NULL, REFERENCES actor.usr.
note : text --
funding_source_credit : integer -- NOT NULL, REFERENCES acq.funding_source_credit.

Indexes:

acqftr_usr_idx : transfer_user

Table: funding_source

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE#1, NOT NULL,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
currency_type : text -- NOT NULL, REFERENCES acq.currency_type.

code : text -- UNIQUE,

Tables referencing acq.fund_allocation via Foreign Key Constraints:

acq.fund_allocation
acq.fund_allocation_percent
acq.funding_source_credit

View: funding_source_allocation_total

Columns:

field name : datatype -- parameters, contraints and notes
funding_source : integer --
amount : numeric(100,2) --

View: funding_source_balance

Columns:

field name : datatype -- parameters, contraints and notes
funding_source : integer --
amount : numeric(100,2) --

Table: funding_source_credit

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
funding_source : integer -- NOT NULL, REFERENCES acq.funding_source.
amount : numeric -- NOT NULL,
note : text --
deadline_date : timestamp with time zone --
effective_date : timestamp with time zone -- NOT NULL, DEFAULT now(),

Tables referencing acq.debit_attribution via Foreign Key Constraints:

acq.debit_attribution
acq.fund_transfer

View: funding_source_credit_total

Columns:

field name : datatype -- parameters, contraints and notes
funding_source : integer --
amount : numeric --

Table: invoice

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
receiver : integer -- NOT NULL, REFERENCES actor.org_unit.
provider : integer -- UNIQUE#1, NOT NULL, REFERENCES acq.provider.
shipper : integer -- NOT NULL, REFERENCES acq.provider.
recv_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
recv_method : text -- NOT NULL, DEFAULT 'EDI'::text, REFERENCES acq.invoice_method.
inv_type : text --
inv_ident : text -- UNIQUE#1, NOT NULL,
payment_auth : text --
payment_method : text -- REFERENCES acq.invoice_payment_method.
note : text --
complete : boolean -- NOT NULL, DEFAULT false,

Tables referencing acq.invoice_entry via Foreign Key Constraints:

acq.invoice_entry acq.invoice_item

Table: invoice_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
invoice : integer -- NOT NULL, REFERENCES acq.invoice.
purchase_order : integer -- REFERENCES acq.purchase_order.
lineitem : integer -- REFERENCES acq.lineitem.
inv_item_count : integer -- NOT NULL,
phys_item_count : integer --
note : text --
billed_per_item : boolean --
cost_billed : numeric(8,2) --
actual_cost : numeric(8,2) --
amount_paid : numeric(8,2) --

Table: invoice_item

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
invoice : integer -- NOT NULL, REFERENCES acq.invoice.
purchase_order : integer -- REFERENCES acq.purchase_order.
fund_debit : integer -- REFERENCES acq.fund_debit.
inv_item_type : text -- NOT NULL, REFERENCES acq.invoice_item_type.
title : text --
author : text --
note : text --
cost_billed : numeric(8,2) --
actual_cost : numeric(8,2) --
fund : integer -- REFERENCES acq.fund.

amount_paid : numeric(8,2) --
po_item : integer -- REFERENCES acq.po_item.
target : bigint --

Table: invoice_item_type

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
name : text -- NOT NULL,
prorate : boolean -- NOT NULL, DEFAULT false,

Tables referencing acq.invoice_item via Foreign Key Constraints:

acq.invoice_item acq.po_item

Table: invoice_method

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
name : text -- NOT NULL,

Tables referencing acq.invoice via Foreign Key Constraints:

acq.invoice

Table: invoice_payment_method

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
name : text -- NOT NULL,

Tables referencing acq.invoice via Foreign Key Constraints:

acq.invoice

Table: lineitem

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
creator : integer -- NOT NULL, REFERENCES actor.usr.
editor : integer -- NOT NULL, REFERENCES actor.usr.
selector : integer -- NOT NULL, REFERENCES actor.usr.

provider : integer -- REFERENCES acq.provider.
purchase_order : integer -- REFERENCES acq.purchase_order.
picklist : integer -- REFERENCES acq.picklist.
expected_recv_time : timestamp with time zone --
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
edit_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
marc : text -- NOT NULL,
eg_bib_id : bigint -- REFERENCES biblio.record_entry.
source_label : text --
state : text -- NOT NULL, DEFAULT 'new'::text,
cancel_reason : integer -- REFERENCES acq.cancel_reason.
estimated_unit_price : numeric --
claim_policy : integer -- REFERENCES acq.claim_policy.

Constraints:

picklist_or_po : CHECK (((picklist IS NOT NULL) OR (purchase_order IS NOT NULL)))

Indexes:

li_creator_idx : creator
li_editor_idx : editor
li_pl_idx : picklist
li_po_idx : purchase_order
li_selector_idx : selector

Tables referencing acq.distribution_formula_application via Foreign Key Constraints:

acq.distribution_formula_application
acq.invoice_entry
acq.lineitem_attr
acq.lineitem_detail
acq.lineitem_note
acq.user_request

Table: lineitem_alert_text

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
code : text -- UNIQUE#1, NOT NULL,
description : text --
owning_lib : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.

Tables referencing acq.lineitem_note via Foreign Key Constraints:

acq.lineitem_note

Table: lineitem_attr

Columns:

field name : datatype -- parameters, contraints and notes

id : bigserial -- PRIMARY KEY,
definition : bigint -- NOT NULL,
lineitem : bigint -- NOT NULL, REFERENCES acq.lineitem.
attr_type : text -- NOT NULL,
attr_name : text -- NOT NULL,
attr_value : text -- NOT NULL,

Indexes:

li_attr_definition_idx : definition
li_attr_li_idx : lineitem
li_attr_value_idx : attr_value

Table: lineitem_attr_definition

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
code : text -- NOT NULL,
description : text -- NOT NULL,
remove : text -- NOT NULL, DEFAULT ''::text,
ident : boolean -- NOT NULL, DEFAULT false,

Table: lineitem_detail

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
lineitem : integer -- NOT NULL, REFERENCES acq.lineitem.
fund : integer -- REFERENCES acq.fund.
fund_debit : integer -- REFERENCES acq.fund_debit.
eg_copy_id : bigint --
barcode : text --
cn_label : text --
note : text --
collection_code : text --
circ_modifier : text -- REFERENCES config.circ_modifier.
owning_lib : integer -- REFERENCES actor.org_unit.
location : integer -- REFERENCES asset.copy_location.
recv_time : timestamp with time zone --
cancel_reason : integer -- REFERENCES acq.cancel_reason.

Indexes:

li_detail_li_idx : lineitem

Tables referencing acq.claim via Foreign Key Constraints:

acq.claim

Table: lineitem_generated_attr_definition

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('acq.lineitem_attr_definition_id_seq'::regclass),
code : text -- NOT NULL,
description : text -- NOT NULL,
remove : text -- NOT NULL, DEFAULT ''::text,
ident : boolean -- NOT NULL, DEFAULT false,
xpath : text -- NOT NULL,

Table: lineitem_local_attr_definition

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('acq.lineitem_attr_definition_id_seq'::regclass),
code : text -- NOT NULL,
description : text -- NOT NULL,
remove : text -- NOT NULL, DEFAULT ''::text,
ident : boolean -- NOT NULL, DEFAULT false,

Table: lineitem_marc_attr_definition

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('acq.lineitem_attr_definition_id_seq'::regclass),
code : text -- NOT NULL,
description : text -- NOT NULL,
remove : text -- NOT NULL, DEFAULT ''::text,
ident : boolean -- NOT NULL, DEFAULT false,
xpath : text -- NOT NULL,

Table: lineitem_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
lineitem : integer -- NOT NULL, REFERENCES acq.lineitem.
creator : integer -- NOT NULL, REFERENCES actor.usr.
editor : integer -- NOT NULL, REFERENCES actor.usr.
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
edit_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

value : text -- NOT NULL,
alert_text : integer -- REFERENCES acq.lineitem_alert_text.
vendor_public : boolean -- NOT NULL, DEFAULT false,

Indexes:

li_note_creator_idx : creator
li_note_editor_idx : editor
li_note_li_idx : lineitem

Table: lineitem_provider_attr_definition

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('acq.lineitem_attr_definition_id_seq'::regclass),
code : text -- NOT NULL,
description : text -- NOT NULL,
remove : text -- NOT NULL, DEFAULT ''::text,
ident : boolean -- NOT NULL, DEFAULT false,
xpath : text -- NOT NULL,
provider : integer -- NOT NULL, REFERENCES acq.provider.

Table: lineitem_usr_attr_definition

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('acq.lineitem_attr_definition_id_seq'::regclass),
code : text -- NOT NULL,
description : text -- NOT NULL,
remove : text -- NOT NULL, DEFAULT ''::text,
ident : boolean -- NOT NULL, DEFAULT false,
usr : integer -- NOT NULL, REFERENCES actor.usr.

Indexes:

li_usr_attr_def_usr_idx : usr

View: ordered_funding_source_credit

The acq.ordered_funding_source_credit view is a prioritized ordering of funding source credits. When ordered by the first three columns, this view defines the order in which the various credits are to be tapped for spending, subject to the allocations in the acq.fund_allocation table. The first column reflects the principle that we should spend money with deadlines before spending money without deadlines. The second column reflects the principle that we should spend the oldest money first. For money with deadlines, that means that we spend first from the credit with the earliest deadline. For money without deadlines, we spend first from the credit with the earliest effective date. The third column is a tie breaker to ensure a consistent ordering.

Columns:

field name : datatype -- parameters, contraints and notes
sort_priority : integer --
sort_date : timestamp with time zone --
id : integer --
funding_source : integer --
amount : numeric --
note : text --

Table: picklist

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
owner : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.usr.
creator : integer -- `NOT NULL`, `REFERENCES` actor.usr.
editor : integer -- `NOT NULL`, `REFERENCES` actor.usr.
org_unit : integer -- `NOT NULL`, `REFERENCES` actor.org_unit.
name : text -- `UNIQUE#1`, `NOT NULL`,
create_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
edit_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,

Indexes:

acq_picklist_creator_idx : creator
acq_picklist_editor_idx : editor
acq_picklist_owner_idx : owner

Tables referencing acq.lineitem via Foreign Key Constraints:

acq.lineitem

Table: po_item

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
purchase_order : integer -- `REFERENCES` acq.purchase_order.
fund_debit : integer -- `REFERENCES` acq.fund_debit.
inv_item_type : text -- `NOT NULL`, `REFERENCES` acq.invoice_item_type.
title : text --
author : text --
note : text --
estimated_cost : numeric(8,2) --
fund : integer -- `REFERENCES` acq.fund.
target : bigint --

Tables referencing acq.invoice_item via Foreign Key Constraints:

acq.invoice_item

Table: po_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
purchase_order : integer -- NOT NULL, REFERENCES acq.purchase_order.
creator : integer -- NOT NULL, REFERENCES actor.usr.
editor : integer -- NOT NULL, REFERENCES actor.usr.
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
edit_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
value : text -- NOT NULL,
vendor_public : boolean -- NOT NULL, DEFAULT false,

Indexes:

acq_po_note_creator_idx : creator
acq_po_note_editor_idx : editor
po_note_po_idx : purchase_order

Table: provider

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE#1, NOT NULL,
owner : integer -- UNIQUE#2, UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
currency_type : text -- NOT NULL, REFERENCES acq.currency_type.
code : text -- UNIQUE#2, NOT NULL,
holding_tag : text --
san : text --
edi_default : integer -- REFERENCES acq.edi_account.
active : boolean -- NOT NULL, DEFAULT true,
prepayment_required : boolean -- NOT NULL, DEFAULT false,
url : text --
email : text --
phone : text --
fax_phone : text --
default_claim_policy : integer -- REFERENCES acq.claim_policy.

Tables referencing acq.edi_account via Foreign Key Constraints:

acq.edi_account
acq.invoice
acq.lineitem
acq.lineitem_provider_attr_definition
acq.provider_address
acq.provider_contact
acq.provider_holding_subfield_map
acq.provider_note
acq.purchase_order

Table: provider_address

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
valid : boolean -- NOT NULL, DEFAULT true,
address_type : text --
provider : integer -- NOT NULL, REFERENCES acq.provider.
street1 : text -- NOT NULL,
street2 : text --
city : text -- NOT NULL,
county : text --
state : text -- NOT NULL,
country : text -- NOT NULL,
post_code : text -- NOT NULL,
fax_phone : text --

Table: provider_contact

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
provider : integer -- NOT NULL, REFERENCES acq.provider.
name : text -- NOT NULL,
role : text --
email : text --
phone : text --

Tables referencing acq.provider_contact_address via Foreign Key Constraints:

acq.provider_contact_address

Table: provider_contact_address

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
valid : boolean -- NOT NULL, DEFAULT true,
address_type : text --
contact : integer -- NOT NULL, REFERENCES acq.provider_contact.
street1 : text -- NOT NULL,
street2 : text --
city : text -- NOT NULL,
county : text --
state : text -- NOT NULL,
country : text -- NOT NULL,
post_code : text -- NOT NULL,

fax_phone : text --

Table: provider_holding_subfield_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
provider : integer -- UNIQUE#1, NOT NULL, REFERENCES acq.provider.
name : text -- UNIQUE#1, NOT NULL,
subfield : text -- NOT NULL,

Table: provider_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
provider : integer -- NOT NULL, REFERENCES acq.provider.
creator : integer -- NOT NULL, REFERENCES actor.usr.
editor : integer -- NOT NULL, REFERENCES actor.usr.
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
edit_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
value : text -- NOT NULL,

Indexes:

acq_pro_note_creator_idx : creator
acq_pro_note_editor_idx : editor
acq_pro_note_pro_idx : provider

Table: purchase_order

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owner : integer -- NOT NULL, REFERENCES actor.usr.
creator : integer -- NOT NULL, REFERENCES actor.usr.
editor : integer -- NOT NULL, REFERENCES actor.usr.
ordering_agency : integer -- NOT NULL, REFERENCES actor.org_unit.
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
edit_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
provider : integer -- NOT NULL, REFERENCES acq.provider.
state : text -- NOT NULL, DEFAULT 'new'::text,
order_date : timestamp with time zone --
name : text -- NOT NULL,
cancel_reason : integer -- REFERENCES acq.cancel_reason.
prepayment_required : boolean -- NOT NULL, DEFAULT false,

Indexes:

acq_po_org_name_order_date_idx : ordering_agency, name, order_date
po_creator_idx : creator
po_editor_idx : editor
po_owner_idx : owner
po_provider_idx : provider
po_state_idx : state

Tables referencing acq.edi_message via Foreign Key Constraints:

acq.edi_message
acq.invoice_entry
acq.invoice_item
acq.lineitem
acq.po_item
acq.po_note

Table: serial_claim

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
type : integer -- `NOT NULL`, `REFERENCES` acq.claim_type.
item : bigint -- `NOT NULL`, `REFERENCES` serial.item.

Indexes:

serial_claim_lid_idx : item

Tables referencing acq.serial_claim_event via Foreign Key Constraints:

acq.serial_claim_event

Table: serial_claim_event

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
type : integer -- `NOT NULL`, `REFERENCES` acq.claim_event_type.
claim : serial -- `NOT NULL`, `REFERENCES` acq.serial_claim.
event_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
creator : integer -- `NOT NULL`, `REFERENCES` actor.usr.
note : text --

Indexes:

serial_claim_event_claim_date_idx : claim, event_date

Table: user_request

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,

usr : integer -- NOT NULL, REFERENCES actor.usr.
hold : boolean -- NOT NULL, DEFAULT true,
pickup_lib : integer -- NOT NULL, REFERENCES actor.org_unit.
holdable_formats : text --
phone_notify : text --
email_notify : boolean -- NOT NULL, DEFAULT true,
lineitem : integer -- REFERENCES acq.lineitem.
eg_bib : bigint -- REFERENCES biblio.record_entry.
request_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
need_before : timestamp with time zone --
max_fee : text --
request_type : integer -- NOT NULL, REFERENCES acq.user_request_type.
isxn : text --
title : text --
volume : text --
author : text --
article_title : text --
article_pages : text --
publisher : text --
location : text --
pubdate : text --
mentioned : text --
other_info : text --
cancel_reason : integer -- REFERENCES acq.cancel_reason.

Table: user_request_type

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
label : text -- UNIQUE, NOT NULL,

Tables referencing acq.user_request via Foreign Key Constraints:

acq.user_request

attribute_debits()

Function Properties

Language: PLPGSQL
Return Type: void

audit_acq_lineitem_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

audit_acq_purchase_order_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

create_acq_auditor(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

create_acq_func(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

create_acq_history(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

create_acq_lifecycle(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

create_acq_seq(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

create_acq_update_trigger(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

exchange_ratio(text, text, numeric)

Function Properties

Language: SQL
Return Type: numeric

exchange_ratio(to_ex text, from_ex text)

Function Properties

Language: PLPGSQL
Return Type: numeric

extract_holding_attr_table(tag integer, lineitem text)

Function Properties

Language: PLPGSQL
Return Type: SET OF flat_lineitem_holding_subfield

extract_provider_holding_data(lineitem_i integer)

Function Properties

Language: PLPGSQL
Return Type: SET OF flat_lineitem_detail

fap_limit_100()

Function Properties

Language: PLPGSQL
Return Type: trigger

find_bad_fy()

Function Properties

Language: PLPGSQL
Return Type: SET OF record

fund_alloc_percent_val()

Function Properties

Language: PLPGSQL
Return Type: trigger

po_org_name_date_unique()

Function Properties

Language: PLPGSQL
Return Type: trigger

propagate_funds_by_org_tree(org_unit_id integer, user_id integer, old_year integer)

Function Properties

Language: PLPGSQL
Return Type: void

propagate_funds_by_org_unit(org_unit_id integer, user_id integer, old_year integer)

Function Properties

Language: PLPGSQL
Return Type: void

purchase_order_name_default()

Function Properties

Language: PLPGSQL
Return Type: trigger

rollover_funds_by_org_tree(org_unit_id integer, user_id integer, old_year integer)

Function Properties

Language: PLPGSQL
Return Type: void

rollover_funds_by_org_unit(org_unit_id integer, user_id integer, old_year integer)

Function Properties

Language: PLPGSQL
Return Type: void

transfer_fund(xfer_note integer, user_id numeric, new_amount integer, new_fund numeric, old_amount integer, old_fund text)

Function Properties

Language: PLPGSQL
Return Type: void

Schema action

Table: aged_circulation

Columns:

field name : datatype -- parameters, contraints and notes
usr_post_code : text --
usr_home_ou : integer -- NOT NULL,
usr_profile : integer -- NOT NULL,
usr_birth_year : integer --
copy_call_number : integer -- NOT NULL,
copy_location : integer -- NOT NULL,
copy_owning_lib : integer -- NOT NULL,
copy_circ_lib : integer -- NOT NULL,
copy_bib_record : bigint -- NOT NULL,
id : bigint -- PRIMARY KEY,
xact_start : timestamp with time zone -- NOT NULL,
xact_finish : timestamp with time zone --
unrecovered : boolean --
target_copy : bigint -- NOT NULL,
circ_lib : integer -- NOT NULL,
circ_staff : integer -- NOT NULL,
checkin_staff : integer --
checkin_lib : integer --
renewal_remaining : integer -- NOT NULL,
due_date : timestamp with time zone --
stop_fines_time : timestamp with time zone --
checkin_time : timestamp with time zone --
create_time : timestamp with time zone -- NOT NULL,
duration : interval --
fine_interval : interval -- NOT NULL,
recurring_fine : numeric(6,2) --
max_fine : numeric(6,2) --
phone_renewal : boolean -- NOT NULL,
desk_renewal : boolean -- NOT NULL,
opac_renewal : boolean -- NOT NULL,
duration_rule : text -- NOT NULL,

recurring_fine_rule : text -- NOT NULL,
max_fine_rule : text -- NOT NULL,
stop_fines : text --
workstation : integer --
checkin_workstation : integer --
checkin_scan_time : timestamp with time zone --
parent_circ : bigint --

Indexes:

action_aged_circulation_target_copy_idx : target_copy
aged_circ_circ_lib_idx : circ_lib
aged_circ_copy_circ_lib_idx : copy_circ_lib
aged_circ_copy_location_idx : copy_location
aged_circ_copy_owning_lib_idx : copy_owning_lib
aged_circ_start_idx : xact_start

View: all_circulation

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
usr_post_code : text --
usr_home_ou : integer --
usr_profile : integer --
usr_birth_year : integer --
copy_call_number : bigint --
copy_location : integer --
copy_owning_lib : integer --
copy_circ_lib : integer --
copy_bib_record : bigint --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
target_copy : bigint --
circ_lib : integer --
circ_staff : integer --
checkin_staff : integer --
checkin_lib : integer --
renewal_remaining : integer --
due_date : timestamp with time zone --
stop_fines_time : timestamp with time zone --
checkin_time : timestamp with time zone --
create_time : timestamp with time zone --
duration : interval --
fine_interval : interval --
recurring_fine : numeric(6,2) --
max_fine : numeric(6,2) --
phone_renewal : boolean --
desk_renewal : boolean --
opac_renewal : boolean --

duration_rule : text --
recurring_fine_rule : text --
max_fine_rule : text --
stop_fines : text --
workstation : integer --
checkin_workstation : integer --
checkin_scan_time : timestamp with time zone --
parent_circ : bigint --

View: billable_circulations

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
usr : integer --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
unrecovered : boolean --
target_copy : bigint --
circ_lib : integer --
circ_staff : integer --
checkin_staff : integer --
checkin_lib : integer --
renewal_remaining : integer --
due_date : timestamp with time zone --
stop_fines_time : timestamp with time zone --
checkin_time : timestamp with time zone --
create_time : timestamp with time zone --
duration : interval --
fine_interval : interval --
recurring_fine : numeric(6,2) --
max_fine : numeric(6,2) --
phone_renewal : boolean --
desk_renewal : boolean --
opac_renewal : boolean --
duration_rule : text --
recurring_fine_rule : text --
max_fine_rule : text --
stop_fines : text --
workstation : integer --
checkin_workstation : integer --
checkin_scan_time : timestamp with time zone --
parent_circ : bigint --

Table: circulation

Columns:

field name : datatype -- parameters, contraints and notes

id : bigint -- PRIMARY KEY, DEFAULT nextval('money.billable_xact_id_seq'::regclass),
usr : integer -- NOT NULL, REFERENCES actor.usr.
xact_start : timestamp with time zone -- NOT NULL, DEFAULT now(),
xact_finish : timestamp with time zone --
unrecovered : boolean --
target_copy : bigint -- NOT NULL,
circ_lib : integer -- NOT NULL, REFERENCES actor.org_unit.
circ_staff : integer -- NOT NULL,
checkin_staff : integer --
checkin_lib : integer --
renewal_remaining : integer -- NOT NULL,
due_date : timestamp with time zone --
stop_fines_time : timestamp with time zone --
checkin_time : timestamp with time zone --
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
duration : interval --
fine_interval : interval -- NOT NULL, DEFAULT '1 day'::interval,
recurring_fine : numeric(6,2) --
max_fine : numeric(6,2) --
phone_renewal : boolean -- NOT NULL, DEFAULT false,
desk_renewal : boolean -- NOT NULL, DEFAULT false,
opac_renewal : boolean -- NOT NULL, DEFAULT false,
duration_rule : text -- NOT NULL,
recurring_fine_rule : text -- NOT NULL,
max_fine_rule : text -- NOT NULL,
stop_fines : text --
workstation : integer -- REFERENCES actor.workstation.
checkin_workstation : integer -- REFERENCES actor.workstation.
checkin_scan_time : timestamp with time zone --
parent_circ : bigint -- REFERENCES action.circulation.

Constraints:

circulation_stop_fines_check : CHECK ((stop_fines = ANY (ARRAY['CHECKIN'::text, 'CLAIMSRETURNED'::text, 'LOST'::text, 'MAXFINES'::text, 'RENEW'::text, 'LONGOVERDUE'::text, 'CLAIMSNEVERCHECKEDOUT'::text])))

Indexes:

action_circulation_target_copy_idx : target_copy
circ_all_usr_idx : usr
circ_checkin_staff_idx : checkin_staff
circ_checkin_time : checkin_time) WHERE (checkin_time IS NOT NULL
circ_circ_lib_idx : circ_lib
circ_circ_staff_idx : circ_staff
circ_open_date_idx : xact_start) WHERE (xact_finish IS NULL
circ_open_xacts_idx : usr) WHERE (xact_finish IS NULL
circ_outstanding_idx : usr) WHERE (checkin_time IS NULL

Tables referencing action.circulation via Foreign Key Constraints:

action.circulation

Table: fieldset

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owner : integer -- NOT NULL, REFERENCES actor.usr.
owning_lib : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
status : text -- NOT NULL,
creation_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
scheduled_time : timestamp with time zone --
applied_time : timestamp with time zone --
classname : text -- NOT NULL,
name : text -- UNIQUE#1, NOT NULL,
stored_query : integer -- REFERENCES query.stored_query.
pkey_value : text --

Constraints:

fieldset_one_or_the_other : CHECK ((((stored_query IS NOT NULL) AND (pkey_value IS NULL)) OR ((pkey_value IS NOT NULL) AND (stored_query IS NULL))))
valid_status : CHECK ((status = ANY (ARRAY['PENDING'::text, 'APPLIED'::text, 'ERROR'::text])))

Indexes:

action_fieldset_sched_time_idx : scheduled_time
action_owner_idx : owner

Tables referencing action.fieldset_col_val via Foreign Key Constraints:

action.fieldset_col_val

Table: fieldset_col_val

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
fieldset : integer -- UNIQUE#1, NOT NULL, REFERENCES action.fieldset.
col : text -- UNIQUE#1, NOT NULL,
val : text --

Table: hold_copy_map

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
hold : integer -- UNIQUE#1, NOT NULL, REFERENCES action.hold_request.
target_copy : bigint -- UNIQUE#1, NOT NULL,

Indexes:

acm_copy_idx : target_copy

Table: hold_notification

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
hold : integer -- NOT NULL, REFERENCES action.hold_request.
notify_staff : integer -- REFERENCES actor.usr.
notify_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
method : text -- NOT NULL,
note : text --

Indexes:

ahn_hold_idx : hold
ahn_notify_staff_idx : notify_staff

Table: hold_request

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
request_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
capture_time : timestamp with time zone --
fulfillment_time : timestamp with time zone --
checkin_time : timestamp with time zone --
return_time : timestamp with time zone --
prev_check_time : timestamp with time zone --
expire_time : timestamp with time zone --
cancel_time : timestamp with time zone --
cancel_cause : integer -- REFERENCES action.hold_request_cancel_cause.
cancel_note : text --
target : bigint -- NOT NULL,
current_copy : bigint --
fulfillment_staff : integer -- REFERENCES actor.usr.
fulfillment_lib : integer -- REFERENCES actor.org_unit.
request_lib : integer -- NOT NULL, REFERENCES actor.org_unit.
requestor : integer -- NOT NULL, REFERENCES actor.usr.
usr : integer -- NOT NULL, REFERENCES actor.usr.
selection_ou : integer -- NOT NULL,
selection_depth : integer -- NOT NULL,
pickup_lib : integer -- NOT NULL, REFERENCES actor.org_unit.
hold_type : text -- NOT NULL,
holdable_formats : text --
phone_notify : text --

email_notify : boolean -- NOT NULL, DEFAULT true,
frozen : boolean -- NOT NULL, DEFAULT false,
thaw_date : timestamp with time zone --
shelf_time : timestamp with time zone --
cut_in_line : boolean --
mint_condition : boolean -- NOT NULL, DEFAULT true,
shelf_expire_time : timestamp with time zone --

Indexes:

hold_request_current_copy_idx : current_copy
hold_request_fulfillment_staff_idx : fulfillment_staff
hold_request_pickup_lib_idx : pickup_lib
hold_request_prev_check_time_idx : prev_check_time
hold_request_requestor_idx : requestor
hold_request_target_idx : target
hold_request_usr_idx : usr

Tables referencing action.hold_copy_map via Foreign Key Constraints:

action.hold_copy_map
action.hold_notification
action.hold_request_note
action.hold_transit_copy

Table: hold_request_cancel_cause

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
label : text -- UNIQUE,

Tables referencing action.hold_request via Foreign Key Constraints:

action.hold_request

Table: hold_request_note

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
hold : bigint -- NOT NULL, REFERENCES action.hold_request.
title : text -- NOT NULL,
body : text -- NOT NULL,
slip : boolean -- NOT NULL, DEFAULT false,
pub : boolean -- NOT NULL, DEFAULT false,
staff : boolean -- NOT NULL, DEFAULT false,

Indexes:

ahrn_hold_idx : hold

Table: hold_transit_copy

Columns:

field name : datatype -- parameters, contraints and notes
id : integer -- PRIMARY KEY, DEFAULT nextval('action.transit_copy_id_seq'::regclass),
source_send_time : timestamp with time zone --
dest_recv_time : timestamp with time zone --
target_copy : bigint -- NOT NULL,
source : integer -- NOT NULL,
dest : integer -- NOT NULL,
prev_hop : integer --
copy_status : integer -- NOT NULL,
persistant_transfer : boolean -- NOT NULL, DEFAULT false,
prev_dest : integer --
hold : integer -- REFERENCES action.hold_request.

Indexes:

active_hold_transit_cp_idx : target_copy
active_hold_transit_dest_idx : dest
active_hold_transit_source_idx : source

Table: in_house_use

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
item : bigint -- NOT NULL,
staff : integer -- NOT NULL, REFERENCES actor.usr.
org_unit : integer -- NOT NULL, REFERENCES actor.org_unit.
use_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Indexes:

action_in_house_use_staff_idx : staff

Table: non_cat_in_house_use

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
item_type : bigint -- NOT NULL, REFERENCES config.non_cataloged_type.
staff : integer -- NOT NULL, REFERENCES actor.usr.
org_unit : integer -- NOT NULL, REFERENCES actor.org_unit.
use_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Indexes:

non_cat_in_house_use_staff_idx : staff

Table: non_cataloged_circulation

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
patron : integer -- NOT NULL, REFERENCES actor.usr.
staff : integer -- NOT NULL, REFERENCES actor.usr.
circ_lib : integer -- NOT NULL, REFERENCES actor.org_unit.
item_type : integer -- NOT NULL, REFERENCES config.non_cataloged_type.
circ_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Indexes:

action_non_cat_circ_patron_idx : patron
action_non_cat_circ_staff_idx : staff

View: open_circulation

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
usr : integer --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
unrecovered : boolean --
target_copy : bigint --
circ_lib : integer --
circ_staff : integer --
checkin_staff : integer --
checkin_lib : integer --
renewal_remaining : integer --
due_date : timestamp with time zone --
stop_fines_time : timestamp with time zone --
checkin_time : timestamp with time zone --
create_time : timestamp with time zone --
duration : interval --
fine_interval : interval --
recurring_fine : numeric(6,2) --
max_fine : numeric(6,2) --
phone_renewal : boolean --
desk_renewal : boolean --
opac_renewal : boolean --
duration_rule : text --
recurring_fine_rule : text --
max_fine_rule : text --
stop_fines : text --
workstation : integer --

checkin_workstation : integer --
checkin_scan_time : timestamp with time zone --
parent_circ : bigint --

Table: reservation_transit_copy

Columns:

field name : datatype -- parameters, contraints and notes
id : integer -- `PRIMARY KEY`, `DEFAULT nextval('action.transit_copy_id_seq'::regclass)`,
source_send_time : timestamp with time zone --
dest_recv_time : timestamp with time zone --
target_copy : bigint -- `NOT NULL`, `REFERENCES` booking.resource.
source : integer -- `NOT NULL`,
dest : integer -- `NOT NULL`,
prev_hop : integer --
copy_status : integer -- `NOT NULL`,
persistant_transfer : boolean -- `NOT NULL`, `DEFAULT false`,
prev_dest : integer --
reservation : integer -- `REFERENCES` booking.reservation.

Indexes:

active_reservation_transit_cp_idx : target_copy
active_reservation_transit_dest_idx : dest
active_reservation_transit_source_idx : source

Table: survey

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
owner : integer -- `NOT NULL`, `REFERENCES` actor.org_unit.
start_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
end_date : timestamp with time zone -- `NOT NULL`, `DEFAULT (now() + '10 years'::interval)`,
usr_summary : boolean -- `NOT NULL`, `DEFAULT false`,
opac : boolean -- `NOT NULL`, `DEFAULT false`,
poll : boolean -- `NOT NULL`, `DEFAULT false`,
required : boolean -- `NOT NULL`, `DEFAULT false`,
name : text -- `NOT NULL`,
description : text -- `NOT NULL`,

Tables referencing action.survey_question via Foreign Key Constraints:

action.survey_question action.survey_response

Table: survey_answer

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
question : integer -- NOT NULL, REFERENCES action.survey_question.
answer : text -- NOT NULL,

Tables referencing action.survey_response via Foreign Key Constraints:

action.survey_response

Table: survey_question

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
survey : integer -- NOT NULL, REFERENCES action.survey.
question : text -- NOT NULL,

Tables referencing action.survey_answer via Foreign Key Constraints:

action.survey_answer

action.survey_response

Table: survey_response

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
response_group_id : integer --
usr : integer --
survey : integer -- NOT NULL, REFERENCES action.survey.
question : integer -- NOT NULL, REFERENCES action.survey_question.
answer : integer -- NOT NULL, REFERENCES action.survey_answer.
answer_date : timestamp with time zone --
effective_date : timestamp with time zone -- NOT NULL, DEFAULT now(),

Indexes:

action_survey_response_usr_idx : usr

Table: transit_copy

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
source_send_time : timestamp with time zone --
dest_recv_time : timestamp with time zone --
target_copy : bigint -- NOT NULL,
source : integer -- NOT NULL, REFERENCES actor.org_unit.
dest : integer -- NOT NULL, REFERENCES actor.org_unit.
prev_hop : integer -- REFERENCES action.transit_copy.

copy_status : integer -- NOT NULL, REFERENCES config.copy_status.
persistant_transfer : boolean -- NOT NULL, DEFAULT false,
prev_dest : integer -- REFERENCES actor.org_unit.

Indexes:

active_transit_cp_idx : target_copy
active_transit_dest_idx : dest
active_transit_source_idx : source

Tables referencing action.transit_copy via Foreign Key Constraints:

action.transit_copy

View: unfulfilled_hold_innermost_loop

Columns:

field name : datatype -- parameters, contraints and notes
hold : integer --
circ_lib : integer --
count : bigint --

Table: unfulfilled_hold_list

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
current_copy : bigint -- NOT NULL,
hold : integer -- NOT NULL,
circ_lib : integer -- NOT NULL,
fail_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Indexes:

uhr_hold_idx : hold

View: unfulfilled_hold_loops

Columns:

field name : datatype -- parameters, contraints and notes
hold : integer --
circ_lib : integer --
count : bigint --

View: unfulfilled_hold_max_loop

Columns:

field name : datatype -- parameters, contraints and notes

hold : integer --
max : bigint --

View: unfulfilled_hold_min_loop

Columns:

field name : datatype -- parameters, contraints and notes
hold : integer --
min : bigint --

age_circ_on_delete()

Function Properties

Language: PLPGSQL
Return Type: trigger

age_parent_circ_on_delete()

Function Properties

Language: PLPGSQL
Return Type: trigger

apply_fieldset(query integer, pkey_name text, table_name text, fieldset_id text)

Applies a specified fieldset, using a supplied table name and primary key name. The query parameter should be non-null only for query-based fieldsets. Returns NULL if successful, or an error message if not.

Function Properties

Language: PLPGSQL
Return Type: text

circ_chain(ctx_circ_id integer)

Function Properties

Language: PLPGSQL
Return Type: SET OF circulation

circulation_claims_returned()

Function Properties

Language: PLPGSQL
Return Type: trigger

copy_related_hold_stats(copy_id integer)

Function Properties

Language: PLPGSQL
Return Type: hold_stats

find_circ_matrix_matchpoint(renewal integer, match_user bigint, match_item integer, context_ou boolean)

Function Properties

Language: PLPGSQL
Return Type: circ_matrix_matchpoint

find_hold_matrix_matchpoint(match_requestor integer, match_user integer, match_item bigint, request_ou integer, pickup_ou integer)

Function Properties

Language: PLPGSQL
Return Type: integer

hold_request_permit_test(match_requestor integer, match_user integer, match_item bigint, request_ou integer, pickup_ou integer)

Function Properties

Language: SQL
Return Type: SET OF matrix_test_result

hold_request_permit_test(retargetting integer, match_requestor integer, match_user bigint, match_item integer, request_ou integer, pickup_ou boolean)

Function Properties

Language: PLPGSQL

Return Type: SET OF matrix_test_result

hold_retarget_permit_test(match_requestor integer, match_user integer, match_item bigint, request_ou integer, pickup_ou integer)

Function Properties

Language: SQL
Return Type: SET OF matrix_test_result

item_user_circ_test(integer, bigint, integer)

Function Properties

Language: SQL
Return Type: SET OF matrix_test_result

item_user_circ_test(renewal integer, match_user bigint, match_item integer, circ_ou boolean)

Function Properties

Language: PLPGSQL
Return Type: SET OF matrix_test_result

item_user_renew_test(integer, bigint, integer)

Function Properties

Language: SQL
Return Type: SET OF matrix_test_result

purge_circulations()

Function Properties

Language: PLPGSQL
Return Type: integer

push_circ_due_time()

Function Properties

Language: PLPGSQL

Return Type: trigger

summarize_circ_chain(ctx_circ_id integer)

Function Properties

Language: PLPGSQL
Return Type: circ_chain_summary

survey_response_answer_date_fixup()

Function Properties

Language: PLPGSQL
Return Type: trigger

usr_visible_circ_copies(integer)

Function Properties

Language: SQL
Return Type: SET OF bigint

usr_visible_circs(usr_id integer)

Function Properties

Language: PLPGSQL
Return Type: SET OF circulation

usr_visible_holds(usr_id integer)

Function Properties

Language: PLPGSQL
Return Type: SET OF hold_request

Schema action_trigger

Table: cleanup

Columns:

field name : datatype -- parameters, contraints and notes
module : text -- `PRIMARY KEY`,

description : text --

Tables referencing action_trigger.event_definition via Foreign Key Constraints:

action_trigger.event_definition

Table: collector

Columns:

field name : datatype -- parameters, contraints and notes
module : text -- PRIMARY KEY,
description : text --

Tables referencing action_trigger.environment via Foreign Key Constraints:

action_trigger.environment

Table: environment

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
event_def : integer -- UNIQUE#1, NOT NULL, REFERENCES action_trigger.event_definition.
path : text --
collector : text -- REFERENCES action_trigger.collector.
label : text -- UNIQUE#1,

Constraints:

environment_label_check : CHECK ((label <> ALL (ARRAY['result'::text, 'target'::text, 'event'::text])))

Table: event

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
target : bigint -- NOT NULL,
event_def : integer -- REFERENCES action_trigger.event_definition.
add_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
run_time : timestamp with time zone -- NOT NULL,
start_time : timestamp with time zone --
update_time : timestamp with time zone --
complete_time : timestamp with time zone --
update_process : integer --
state : text -- NOT NULL, DEFAULT 'pending'::text,
user_data : text --
template_output : bigint -- REFERENCES action_trigger.event_output.

error_output : bigint -- REFERENCES action_trigger.event_output.
async_output : bigint -- REFERENCES action_trigger.event_output.

Constraints:

event_state_check : CHECK ((state = ANY (ARRAY['pending'::text, 'invalid'::text, 'found'::text, 'collecting'::text, 'collected'::text, 'validating'::text, 'valid'::text, 'reacting'::text, 'reacted'::text, 'cleaning'::text, 'complete'::text, 'error'::text])))
event_user_data_check : CHECK (((user_data IS NULL) OR is_json(user_data)))

Indexes:

atev_target_def_idx : target, event_def

Table: event_definition

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
active : boolean -- NOT NULL, DEFAULT true,
owner : integer -- UNIQUE#2, UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
name : text -- UNIQUE#2, NOT NULL,
hook : text -- UNIQUE#1, NOT NULL, REFERENCES action_trigger.hook.
validator : text -- UNIQUE#1, NOT NULL, REFERENCES action_trigger.validator.
reactor : text -- UNIQUE#1, NOT NULL, REFERENCES action_trigger.reactor.
cleanup_success : text -- REFERENCES action_trigger.cleanup.
cleanup_failure : text -- REFERENCES action_trigger.cleanup.
delay : interval -- UNIQUE#1, NOT NULL, DEFAULT '00:05:00'::interval,
max_delay : interval --
usr_field : text --
opt_in_setting : text -- REFERENCES config.usr_setting_type.
delay_field : text -- UNIQUE#1,
group_field : text --
template : text --
granularity : text --

Tables referencing action_trigger.environment via Foreign Key Constraints:

action_trigger.environment
action_trigger.event
action_trigger.event_params

Table: event_output

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
is_error : boolean -- NOT NULL, DEFAULT false,

data : text -- NOT NULL,

Tables referencing action_trigger.event via Foreign Key Constraints:

action_trigger.event

Table: event_params

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
event_def : integer -- UNIQUE#1, NOT NULL, REFERENCES action_trigger.event_definition.
param : text -- UNIQUE#1, NOT NULL,
value : text -- NOT NULL,

Table: hook

Columns:

field name : datatype -- parameters, contraints and notes
key : text -- PRIMARY KEY,
core_type : text -- NOT NULL,
description : text --
passive : boolean -- NOT NULL, DEFAULT false,

Tables referencing action_trigger.event_definition via Foreign Key Constraints:

action_trigger.event_definition

Table: reactor

Columns:

field name : datatype -- parameters, contraints and notes
module : text -- PRIMARY KEY,
description : text --

Tables referencing action_trigger.event_definition via Foreign Key Constraints:

action_trigger.event_definition

Table: validator

Columns:

field name : datatype -- parameters, contraints and notes
module : text -- PRIMARY KEY,
description : text --

Tables referencing action_trigger.event_definition via Foreign Key Constraints:

action_trigger.event_definition

Schema actor

Holds all tables pertaining to users and libraries (org units).

Table: card

Library Cards Each User has one or more library cards. The current "main" card is linked to here from the actor.usr table, and it is up to the consortium policy whether more than one card can be active for any one user at a given time.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
usr : integer -- `NOT NULL`, `REFERENCES` actor.usr.
barcode : text -- `UNIQUE`, `NOT NULL`,
active : boolean -- `NOT NULL`, `DEFAULT true`,

Indexes:

actor_card_barcode_evergreen_lowercase_idx : lowercase(barcode)
actor_card_usr_idx : usr

Table: hours_of_operation

When does this org_unit usually open and close? (Variations are expressed in the actor.org_unit_closed table.)

Columns:

field name : datatype -- parameters, contraints and notes
id : integer -- `PRIMARY KEY`, `REFERENCES` actor.org_unit.
dow_0_open : time without time zone -- `NOT NULL`, `DEFAULT '09:00:00'::time without time zone`, When does this org_unit open on Monday?
dow_0_close : time without time zone -- `NOT NULL`, `DEFAULT '17:00:00'::time without time zone`, When does this org_unit close on Monday?
dow_1_open : time without time zone -- `NOT NULL`, `DEFAULT '09:00:00'::time without time zone`, When does this org_unit open on Tuesday?
dow_1_close : time without time zone -- `NOT NULL`, `DEFAULT '17:00:00'::time without time zone`, When does this org_unit close on Tuesday?
dow_2_open : time without time zone -- `NOT NULL`, `DEFAULT '09:00:00'::time without time zone`, When does this org_unit open on Wednesday?
dow_2_close : time without time zone -- `NOT NULL`, `DEFAULT '17:00:00'::time without time zone`, When does this org_unit close on Wednesday?
dow_3_open : time without time zone -- `NOT NULL`, `DEFAULT '09:00:00'::time without time zone`, When does this org_unit open on Thursday?
dow_3_close : time without time zone -- `NOT NULL`, `DEFAULT '17:00:00'::time without time zone`, When does this org_unit close on Thursday?

dow_4_open : time without time zone -- NOT NULL, DEFAULT '09:00:00'::time without time zone, When does this org_unit open on Friday?
dow_4_close : time without time zone -- NOT NULL, DEFAULT '17:00:00'::time without time zone, When does this org_unit close on Friday?
dow_5_open : time without time zone -- NOT NULL, DEFAULT '09:00:00'::time without time zone, When does this org_unit open on Saturday?
dow_5_close : time without time zone -- NOT NULL, DEFAULT '17:00:00'::time without time zone, When does this org_unit close on Saturday?
dow_6_open : time without time zone -- NOT NULL, DEFAULT '09:00:00'::time without time zone, When does this org_unit open on Sunday?
dow_6_close : time without time zone -- NOT NULL, DEFAULT '17:00:00'::time without time zone, When does this org_unit close on Sunday?

Table: org_address

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
valid : boolean -- NOT NULL, DEFAULT true,
address_type : text -- NOT NULL, DEFAULT 'MAILING'::text,
org_unit : integer -- NOT NULL, REFERENCES actor.org_unit.
street1 : text -- NOT NULL,
street2 : text --
city : text -- NOT NULL,
county : text --
state : text -- NOT NULL,
country : text -- NOT NULL,
post_code : text -- NOT NULL,
san : text --

Indexes:

actor_org_address_org_unit_idx : org_unit

Tables referencing actor.org_unit via Foreign Key Constraints:

actor.org_unit

Table: org_lasso

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE,

Tables referencing actor.org_lasso_map via Foreign Key Constraints:

actor.org_lasso_map

Table: org_lasso_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
lasso : integer -- NOT NULL, REFERENCES actor.org_lasso.
org_unit : integer -- NOT NULL, REFERENCES actor.org_unit.

Indexes:

ou_lasso_org_unit_idx : org_unit

Table: org_unit

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
parent_ou : integer -- REFERENCES actor.org_unit.
ou_type : integer -- NOT NULL, REFERENCES actor.org_unit_type.
ill_address : integer -- REFERENCES actor.org_address.
holds_address : integer -- REFERENCES actor.org_address.
mailing_address : integer -- REFERENCES actor.org_address.
billing_address : integer -- REFERENCES actor.org_address.
shortname : text -- UNIQUE, NOT NULL,
name : text -- UNIQUE, NOT NULL,
email : text --
phone : text --
opac_visible : boolean -- NOT NULL, DEFAULT true,
fiscal_calendar : integer -- NOT NULL, DEFAULT 1, REFERENCES acq.fiscal_calendar.

Indexes:

actor_org_unit_billing_address_idx : billing_address
actor_org_unit_holds_address_idx : holds_address
actor_org_unit_ill_address_idx : ill_address
actor_org_unit_mailing_address_idx : mailing_address
actor_org_unit_ou_type_idx : ou_type
actor_org_unit_parent_ou_idx : parent_ou

Tables referencing acq.cancel_reason via Foreign Key Constraints:

acq.cancel_reason
acq.claim_policy
acq.distribution_formula
acq.fund
acq.fund_tag
acq.invoice
acq.lineitem_detail
acq.claim_event_type
acq.claim_type
acq.distribution_formula_entry
acq.fund_allocation_percent
acq.funding_source
acq.lineitem_alert_text
acq.picklist

acq.provider
acq.purchase_order
acq.user_request
action.circulation
action.fieldset
action.hold_request
action.in_house_use
action.non_cat_in_house_use
action.non_cataloged_circulation
action.survey
action.transit_copy
action_trigger.event_definition
actor.hours_of_operation
actor.org_address
actor.org_lasso_map
actor.org_unit
actor.org_unit_closed
actor.org_unit_setting
actor.stat_cat
actor.stat_cat_entry
actor.usr
actor.usr_org_unit_opt_in
actor.usr_standing_penalty
actor.workstation
asset.call_number
asset.copy
asset.copy_location
asset.copy_location_order
asset.copy_template
asset.stat_cat
asset.stat_cat_entry
biblio.record_entry
booking.reservation
booking.resource
booking.resource_attr
booking.resource_attr_value
booking.resource_type
config.billing_type
config.circ_matrix_matchpoint
config.hold_matrix_matchpoint
config.idl_field_doc
config.remote_account
money.collections_tracker
permission.grp_penalty_threshold
permission.usr_work_ou_map
reporter.output_folder
reporter.report_folder
reporter.template_folder
serial.distribution
serial.record_entry
serial.subscription
vandelay.import_bib_trash_fields
vandelay.import_item_attr_definition
vandelay.merge_profile

Table: org_unit_closed

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
org_unit : integer -- `NOT NULL`, `REFERENCES` actor.org_unit.
close_start : timestamp with time zone -- `NOT NULL`,
close_end : timestamp with time zone -- `NOT NULL`,
reason : text --

Table: org_unit_proximity

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
from_org : integer --
to_org : integer --
prox : integer --

Indexes:

from_prox_idx : from_org

Table: org_unit_setting

Org Unit settings This table contains any arbitrary settings that a client program would like to save for an org unit.

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
org_unit : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.org_unit.
name : text -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` config.org_unit_setting_type.
value : text -- `NOT NULL`,

Indexes:

actor_org_unit_setting_usr_idx : org_unit

Table: org_unit_type

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
name : text -- `NOT NULL`,
opac_label : text -- `NOT NULL`,
depth : integer -- `NOT NULL`,
parent : integer -- `REFERENCES` actor.org_unit_type.
can_have_vols : boolean -- `NOT NULL`, `DEFAULT true`,
can_have_users : boolean -- `NOT NULL`, `DEFAULT true`,

Indexes:

actor_org_unit_type_parent_idx : parent

Tables referencing actor.org_unit via Foreign Key Constraints:

actor.org_unit
actor.org_unit_type
config.hold_matrix_matchpoint

Table: stat_cat

User Statistical Catagories Local data collected about Users is placed into a Statistical Catagory. Here's where those catagories are defined.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
owner : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.org_unit.

name : text -- UNIQUE#1, NOT NULL,
opac_visible : boolean -- NOT NULL, DEFAULT false,
usr_summary : boolean -- NOT NULL, DEFAULT false,

Tables referencing actor.stat_cat_entry via Foreign Key Constraints:

actor.stat_cat_entry

actor.stat_cat_entry_usr_map

Table: stat_cat_entry

User Statistical Catagory Entries Local data collected about Users is placed into a Statistical Catagory. Each library can create entries into any of its own stat_cats, its ancestors' stat_cats, or its descendants' stat_cats.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
stat_cat : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.stat_cat.
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
value : text -- UNIQUE#1, NOT NULL,

Table: stat_cat_entry_usr_map

Statistical Catagory Entry to User map Records the stat_cat entries for each user.

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
stat_cat_entry : text -- NOT NULL,
stat_cat : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.stat_cat.
target_usr : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.usr.

Indexes:

actor_stat_cat_entry_usr_idx : target_usr

Table: usr

User objects This table contains the core User objects that describe both staff members and patrons. The difference between the two types of users is based on the user's permissions.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
card : integer -- UNIQUE,
profile : integer -- NOT NULL, REFERENCES permission.grp_tree.
usrname : text -- UNIQUE, NOT NULL,
email : text --

passwd : text -- NOT NULL,
standing : integer -- NOT NULL, DEFAULT 1, REFERENCES config.standing.
ident_type : integer -- NOT NULL, REFERENCES config.identification_type.
ident_value : text --
ident_type2 : integer -- REFERENCES config.identification_type.
ident_value2 : text --
net_access_level : integer -- NOT NULL, DEFAULT 1, REFERENCES config.net_access_level.
photo_url : text --
prefix : text --
first_given_name : text -- NOT NULL,
second_given_name : text --
family_name : text -- NOT NULL,
suffix : text --
alias : text --
day_phone : text --
evening_phone : text --
other_phone : text --
mailing_address : integer -- REFERENCES actor.usr_address.
billing_address : integer -- REFERENCES actor.usr_address.
home_ou : integer -- NOT NULL, REFERENCES actor.org_unit.
dob : timestamp with time zone --
active : boolean -- NOT NULL, DEFAULT true,
master_account : boolean -- NOT NULL, DEFAULT false,
super_user : boolean -- NOT NULL, DEFAULT false,
barred : boolean -- NOT NULL, DEFAULT false,
deleted : boolean -- NOT NULL, DEFAULT false,
juvenile : boolean -- NOT NULL, DEFAULT false,
usrgroup : serial -- NOT NULL,
claims_returned_count : integer -- NOT NULL,
credit_forward_balance : numeric(6,2) -- NOT NULL, DEFAULT 0.00,
last_xact_id : text -- NOT NULL, DEFAULT 'none'::text,
alert_message : text --
create_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
expire_date : timestamp with time zone -- NOT NULL, DEFAULT (now() + '3 years'::interval),
claims_never_checked_out_count : integer -- NOT NULL,

Indexes:

actor_usr_billing_address_idx : billing_address
actor_usr_day_phone_idx : lowercase(day_phone)
actor_usr_email_idx : lowercase(email)
actor_usr_evening_phone_idx : lowercase(evening_phone)
actor_usr_family_name_idx : lowercase(family_name)
actor_usr_first_given_name_idx : lowercase(first_given_name)
actor_usr_home_ou_idx : home_ou
actor_usr_ident_value2_idx : lowercase(ident_value2)
actor_usr_ident_value_idx : lowercase(ident_value)
actor_usr_mailing_address_idx : mailing_address
actor_usr_other_phone_idx : lowercase(other_phone)
actor_usr_second_given_name_idx : lowercase(second_given_name)
actor_usr_usrgroup_idx : usrgroup

Tables referencing acq.claim_event via Foreign Key Constraints:

acq.claim_event
acq.distribution_formula_application
acq.fund_allocation
acq.fund_allocation_percent
acq.fund_transfer
acq.lineitem
acq.lineitem_note
acq.lineitem_usr_attr_definition
acq.picklist
acq.po_note
acq.provider_note
acq.purchase_order
acq.serial_claim_event
acq.user_request
action.circulation
action.fieldset
action.hold_notification
action.hold_request
action.in_house_use
action.non_cat_in_house_use
action.non_cataloged_circulation
actor.card
actor.stat_cat_entry_usr_map
actor.usr_address
actor.usr_note
actor.usr_org_unit_opt_in
actor.usr_password_reset
actor.usr_saved_search
actor.usr_setting
actor.usr_standing_penalty
asset.call_number
asset.call_number_note
asset.copy
asset.copy_note
asset.copy_template
biblio.record_entry
biblio.record_note
booking.reservation
container.biblio_record_entry_bucket
container.call_number_bucket
container.copy_bucket
container.user_bucket
container.user_bucket_item
money.billable_xact
money.collections_tracker
permission.usr_grp_map
permission.usr_object_perm_map
permission.usr_perm_map
permission.usr_work_ou_map
reporter.output_folder
reporter.report
reporter.report_folder
reporter.schedule
reporter.template
reporter.template_folder
serial.distribution_note
serial.issuance
serial.item
serial.item_note
serial.routing_list_user
serial.subscription_note
serial.unit
vandelay.queue

Table: usr_address

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
valid : boolean -- `NOT NULL`, `DEFAULT true`,
within_city_limits : boolean -- `NOT NULL`, `DEFAULT true`,
address_type : text -- `NOT NULL`, `DEFAULT 'MAILING'::text`,
usr : integer -- `NOT NULL`, `REFERENCES` actor.usr.
street1 : text -- `NOT NULL`,
street2 : text --
city : text -- `NOT NULL`,
county : text --
state : text -- `NOT NULL`,
country : text -- `NOT NULL`,

post_code : text -- NOT NULL,
pending : boolean -- NOT NULL, DEFAULT false,
replaces : integer -- REFERENCES actor.usr_address.

Indexes:

actor_usr_addr_city_idx : lowercase(city)
actor_usr_addr_post_code_idx : lowercase(post_code)
actor_usr_addr_state_idx : lowercase(state)
actor_usr_addr_street1_idx : lowercase(street1)
actor_usr_addr_street2_idx : lowercase(street2)
actor_usr_addr_usr_idx : usr

Tables referencing actor.usr via Foreign Key Constraints:

actor.usr actor.usr_address

Table: usr_note

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
usr : bigint -- NOT NULL, REFERENCES actor.usr.
creator : bigint -- NOT NULL, REFERENCES actor.usr.
create_date : timestamp with time zone -- DEFAULT now(),
pub : boolean -- NOT NULL, DEFAULT false,
title : text -- NOT NULL,
value : text -- NOT NULL,

Indexes:

actor_usr_note_creator_idx : creator
actor_usr_note_usr_idx : usr

Table: usr_org_unit_opt_in

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
org_unit : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
usr : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.usr.
staff : integer -- NOT NULL, REFERENCES actor.usr.
opt_in_ts : timestamp with time zone -- NOT NULL, DEFAULT now(),
opt_in_ws : integer -- NOT NULL, REFERENCES actor.workstation.

Indexes:

usr_org_unit_opt_in_staff_idx : staff

Table: usr_password_reset

Self-serve password reset requests

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
uuid : text -- NOT NULL,
usr : bigint -- NOT NULL, REFERENCES actor.usr.
request_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
has_been_reset : boolean -- NOT NULL, DEFAULT false,

Indexes:

actor_usr_password_reset_has_been_reset_idx : has_been_reset
actor_usr_password_reset_request_time_idx : request_time
actor_usr_password_reset_usr_idx : usr

Table: usr_saved_search

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.usr.
name : text -- UNIQUE#1, NOT NULL,
create_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
query_text : text -- NOT NULL,
query_type : text -- NOT NULL, DEFAULT 'URL'::text,
target : text -- NOT NULL,

Constraints:

valid_query_text : CHECK ((query_type = 'URL'::text))
valid_target : CHECK ((target = ANY (ARRAY['record'::text, 'metarecord'::text, 'callnumber'::text])))

Table: usr_setting

User settings This table contains any arbitrary settings that a client program would like to save for a user.

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
usr : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.usr.
name : text -- UNIQUE#1, NOT NULL, REFERENCES config.usr_setting_type.
value : text -- NOT NULL,

Indexes:

actor_usr_setting_usr_idx : usr

Table: usr_standing_penalty

User standing penalties

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
org_unit : integer -- `NOT NULL`, `REFERENCES` actor.org_unit.
usr : integer -- `NOT NULL`, `REFERENCES` actor.usr.
standing_penalty : integer -- `NOT NULL`, `REFERENCES` config.standing_penalty.
staff : integer -- `REFERENCES` actor.usr.
set_date : timestamp with time zone -- `DEFAULT now()`,
stop_date : timestamp with time zone --
note : text --

Indexes:

actor_usr_standing_penalty_staff_idx : staff
actor_usr_standing_penalty_usr_idx : usr

Table: workstation

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
name : text -- `UNIQUE`, `NOT NULL`,
owning_lib : integer -- `NOT NULL`, `REFERENCES` actor.org_unit.

Tables referencing action.circulation via Foreign Key Constraints:

action.circulation
actor.usr_org_unit_opt_in
money.bnm_desk_payment

approve_pending_address(pending_id integer)

Replaces an address with a pending address. This is done by giving the pending address the ID of the old address. The replaced address is retained with -id.

Function Properties

Language: PLPGSQL
Return Type: bigint

calculate_system_penalties(context_org integer, match_user integer)

Function Properties

Language: PLPGSQL

Return Type: SET OF usr_standing_penalty

crypt_pw_insert()

Function Properties

Language: PLPGSQL
Return Type: trigger

crypt_pw_update()

Function Properties

Language: PLPGSQL
Return Type: trigger

org_unit_ancestor_at_depth(integer, integer)

Function Properties

Language: SQL
Return Type: org_unit

org_unit_ancestor_setting(org_id text, setting_name integer)

Search "up" the org_unit tree until we find the first occurrence of an org_unit_setting with the given name.

Function Properties

Language: PLPGSQL
Return Type: SET OF org_unit_setting

org_unit_ancestors(integer)

Function Properties

Language: SQL
Return Type: SET OF org_unit

org_unit_ancestors_distance(distance integer)

Function Properties

Language: SQL
Return Type: SET OF record

org_unit_combined_ancestors(integer, integer)

Function Properties

Language: SQL
Return Type: SET OF org_unit

org_unit_common_ancestors(integer, integer)

Function Properties

Language: SQL
Return Type: SET OF org_unit

org_unit_descendants(integer)

Function Properties

Language: SQL
Return Type: SET OF org_unit

org_unit_descendants(integer, integer)

Function Properties

Language: SQL
Return Type: SET OF org_unit

org_unit_descendants_distance(distance integer)

Function Properties

Language: SQL
Return Type: SET OF record

org_unit_full_path(integer)

Function Properties

Language: SQL
Return Type: SET OF org_unit

org_unit_full_path(integer, integer)

Function Properties

Language: SQL

Return Type: SET OF org_unit

org_unit_proximity(integer, integer)

Function Properties

Language: SQL
Return Type: integer

usr_delete(dest_usr integer, src_usr integer)

Logically deletes a user. Removes personally identifiable information, and purges associated data in other tables.

Function Properties

Language: PLPGSQL
Return Type: void

usr_merge(deactivate_cards integer, del_cards integer, del_addrs boolean, dest_usr boolean, src_usr boolean)

Merges all user date from src_usr to dest_usr. When collisions occur, keep dest_usr's data and delete src_usr's data.

Function Properties

Language: PLPGSQL
Return Type: void

usr_merge_rows(dest_usr text, src_usr text, col_name integer, table_name integer)

Attempts to move each row of the specified table from src_user to dest_user. Where conflicts exist, the conflicting "source" row is deleted.

Function Properties

Language: PLPGSQL
Return Type: void

usr_purge_data(specified_dest_usr integer, src_usr integer)

Finds rows dependent on a given row in actor.usr and either deletes them or reassigns them to a different user.

Function Properties

Language: PLPGSQL
Return Type: void

Schema asset

Table: call_number

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
creator : bigint -- `NOT NULL`, `REFERENCES` actor.usr.
create_date : timestamp with time zone -- `DEFAULT now()`,
editor : bigint -- `NOT NULL`, `REFERENCES` actor.usr.
edit_date : timestamp with time zone -- `DEFAULT now()`,
record : bigint -- `NOT NULL`, `REFERENCES` biblio.record_entry.
owning_lib : integer -- `NOT NULL`, `REFERENCES` actor.org_unit.
label : text -- `NOT NULL`,
deleted : boolean -- `NOT NULL`, `DEFAULT false`,
label_class : bigint -- `NOT NULL`, `DEFAULT 1`, `REFERENCES` asset.call_number_class.
label_sortkey : text --

Indexes:

asset_call_number_creator_idx : creator
asset_call_number_dewey_idx : call_number_dewey(label)
asset_call_number_editor_idx : editor
asset_call_number_label_sortkey : oils_text_as_bytea(label_sortkey)
asset_call_number_label_sortkey_browse : oils_text_as_bytea(label_sortkey), oils_text_as_bytea(label), id, owning_lib) WHERE ((deleted IS FALSE) OR (deleted = false)
asset_call_number_record_idx : record
asset_call_number_upper_label_id_owning_lib_idx : oils_text_as_bytea(label), id, owning_lib

Tables referencing asset.call_number_note via Foreign Key Constraints:

asset.call_number_note
asset.copy
asset.uri_call_number_map
container.call_number_bucket_item
serial.distribution
serial.unit

Table: call_number_class

Defines the call number normalization database functions in the "normalizer" column and the tag/subfield combinations to use to lookup the call number in the "field" column for a given classification scheme. Tag/subfield combinations are delimited by commas.

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
name : text -- `NOT NULL`,
normalizer : text -- `NOT NULL`, `DEFAULT 'asset.normalize_generic'::text`,
field : text -- `NOT NULL`, `DEFAULT '050ab,055ab,060ab,070ab,080ab,082ab,086ab,088ab,090,092,096,098,099'::text`,

Tables referencing asset.call_number via Foreign Key Constraints:

asset.call_number

Table: call_number_note

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
call_number : bigint -- NOT NULL, REFERENCES asset.call_number.
creator : bigint -- NOT NULL, REFERENCES actor.usr.
create_date : timestamp with time zone -- DEFAULT now(),
pub : boolean -- NOT NULL, DEFAULT false,
title : text -- NOT NULL,
value : text -- NOT NULL,

Indexes:

asset_call_number_note_creator_idx : creator

Table: copy

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
circ_lib : integer -- NOT NULL, REFERENCES actor.org_unit.
creator : bigint -- NOT NULL, REFERENCES actor.usr.
call_number : bigint -- NOT NULL, REFERENCES asset.call_number.
editor : bigint -- NOT NULL, REFERENCES actor.usr.
create_date : timestamp with time zone -- DEFAULT now(),
edit_date : timestamp with time zone -- DEFAULT now(),
copy_number : integer --
status : integer -- NOT NULL, REFERENCES config.copy_status.
location : integer -- NOT NULL, DEFAULT 1, REFERENCES asset.copy_location.
loan_duration : integer -- NOT NULL,
fine_level : integer -- NOT NULL,
age_protect : integer --
circulate : boolean -- NOT NULL, DEFAULT true,
deposit : boolean -- NOT NULL, DEFAULT false,
ref : boolean -- NOT NULL, DEFAULT false,
holdable : boolean -- NOT NULL, DEFAULT true,
deposit_amount : numeric(6,2) -- NOT NULL, DEFAULT 0.00,
price : numeric(8,2) --
barcode : text -- NOT NULL,
circ_modifier : text -- REFERENCES config.circ_modifier.
circ_as_type : text --
dummy_title : text --
dummy_author : text --
alert_message : text --

opac_visible : boolean -- NOT NULL, DEFAULT true,
deleted : boolean -- NOT NULL, DEFAULT false,
floating : boolean -- NOT NULL, DEFAULT false,
dummy_isbn : text --
status_changed_time : timestamp with time zone --
mint_condition : boolean -- NOT NULL, DEFAULT true,
cost : numeric(8,2) --

Constraints:

copy_fine_level_check : CHECK ((fine_level = ANY (ARRAY[1, 2, 3])))
copy_loan_duration_check : CHECK ((loan_duration = ANY (ARRAY[1, 2, 3])))

Indexes:

cp_avail_cn_idx : call_number
cp_cn_idx : call_number
cp_create_date : create_date
cp_creator_idx : creator
cp_editor_idx : editor

Tables referencing asset.copy_note via Foreign Key Constraints:

asset.copy_note
container.copy_bucket_item

Table: copy_location

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE#1, NOT NULL,
owning_lib : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
holdable : boolean -- NOT NULL, DEFAULT true,
hold_verify : boolean -- NOT NULL, DEFAULT false,
opac_visible : boolean -- NOT NULL, DEFAULT true,
circulate : boolean -- NOT NULL, DEFAULT true,
label_prefix : text --
label_suffix : text --

Tables referencing acq.distribution_formula_entry via Foreign Key Constraints:

acq.distribution_formula_entry
acq.lineitem_detail
asset.copy
asset.copy_location_order
asset.copy_template

Table: copy_location_order

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
location : integer -- UNIQUE#1, NOT NULL, REFERENCES asset.copy_location.

org : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
position : integer -- NOT NULL,

Table: copy_note

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
owning_copy : bigint -- NOT NULL, REFERENCES asset.copy.
creator : bigint -- NOT NULL, REFERENCES actor.usr.
create_date : timestamp with time zone -- DEFAULT now(),
pub : boolean -- NOT NULL, DEFAULT false,
title : text -- NOT NULL,
value : text -- NOT NULL,

Indexes:

asset_copy_note_creator_idx : creator
asset_copy_note_owning_copy_idx : owning_copy

Table: copy_template

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owning_lib : integer -- NOT NULL, REFERENCES actor.org_unit.
creator : bigint -- NOT NULL, REFERENCES actor.usr.
editor : bigint -- NOT NULL, REFERENCES actor.usr.
create_date : timestamp with time zone -- DEFAULT now(),
edit_date : timestamp with time zone -- DEFAULT now(),
name : text -- NOT NULL,
circ_lib : integer -- REFERENCES actor.org_unit.
status : integer -- REFERENCES config.copy_status.
location : integer -- REFERENCES asset.copy_location.
loan_duration : integer --
fine_level : integer --
age_protect : integer --
circulate : boolean --
deposit : boolean --
ref : boolean --
holdable : boolean --
deposit_amount : numeric(6,2) --
price : numeric(8,2) --
circ_modifier : text --
circ_as_type : text --
alert_message : text --
opac_visible : boolean --
floating : boolean --
mint_condition : boolean --

Constraints:

valid_fine_level : CHECK (((fine_level IS NULL) OR (loan_duration = ANY (ARRAY[1, 2, 3]))))
valid_loan_duration : CHECK (((loan_duration IS NULL) OR (loan_duration = ANY (ARRAY[1, 2, 3]))))

Tables referencing serial.distribution via Foreign Key Constraints:

serial.distribution

Table: opac_visible_copies

Materialized view of copies that are visible in the OPAC, used by search.query_parser_fts() to speed up OPAC visibility checks on large databases. Contents are maintained by a set of triggers.

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`,
record : bigint --
circ_lib : integer --

Indexes:

opac_visible_copies_idx1 : record, circ_lib

Table: stat_cat

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
owner : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.org_unit.
opac_visible : boolean -- `NOT NULL`, `DEFAULT false`,
name : text -- `UNIQUE#1`, `NOT NULL`,
required : boolean -- `NOT NULL`, `DEFAULT false`,

Tables referencing asset.stat_cat_entry via Foreign Key Constraints:

asset.stat_cat_entry

asset.stat_cat_entry_copy_map

Table: stat_cat_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
stat_cat : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` asset.stat_cat.
owner : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.org_unit.
value : text -- `UNIQUE#1`, `NOT NULL`,

Tables referencing asset.stat_cat_entry_copy_map via Foreign Key Constraints:

asset.stat_cat_entry_copy_map

Table: stat_cat_entry_copy_map

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
stat_cat : integer -- UNIQUE#1, NOT NULL, REFERENCES asset.stat_cat.
stat_cat_entry : integer -- NOT NULL, REFERENCES asset.stat_cat_entry.
owning_copy : bigint -- UNIQUE#1, NOT NULL,

Indexes:

scecm_owning_copy_idx : owning_copy

Table: stat_cat_entry_transparency_map

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
stat_cat : integer -- UNIQUE#1, NOT NULL,
stat_cat_entry : integer -- NOT NULL,
owning_transparency : integer -- UNIQUE#1, NOT NULL,

Table: uri

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
href : text -- NOT NULL,
label : text --
use_restriction : text --
active : boolean -- NOT NULL, DEFAULT true,

Tables referencing asset.uri_call_number_map via Foreign Key Constraints:

asset.uri_call_number_map serial.item

Table: uri_call_number_map

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
uri : integer -- UNIQUE#1, NOT NULL, REFERENCES asset.uri.
call_number : integer -- UNIQUE#1, NOT NULL, REFERENCES asset.call_number.

Indexes:

asset_uri_call_number_map_cn_idx : call_number

acp_status_changed()

Function Properties

Language: PLPGSQL
Return Type: trigger

autogenerate_placeholder_barcode()

Function Properties

Language: PLPGSQL
Return Type: trigger

cache_copy_visibility()

Trigger function to update the copy OPAC visiblity cache.

Function Properties

Language: PLPGSQL
Return Type: trigger

label_normalizer()

Function Properties

Language: PLPGSQL
Return Type: trigger

label_normalizer_dewey(text)

Function Properties

Language: PLPERLU
Return Type: text

label_normalizer_generic(text)

Function Properties

Language: PLPERLU
Return Type: text

label_normalizer_lc(text)

Function Properties

Language: PLPERLU
Return Type: text

merge_record_assets(source_record bigint, target_record bigint)

Function Properties

Language: PLPGSQL
Return Type: integer

metarecord_copy_count(transcendant integer, unshadow bigint, available boolean)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

opac_lasso_metarecord_copy_count(transcendant integer, unshadow bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

opac_lasso_record_copy_count(transcendant integer, unshadow bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

opac_ou_metarecord_copy_count(transcendant integer, unshadow bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

opac_ou_record_copy_count(transcendant integer, unshadow bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

record_copy_count(transcendant integer, unshadow bigint, available boolean)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

refresh_opac_visible_copies_mat_view()

Rebuild the copy OPAC visibility cache. Useful during migrations.

Function Properties

Language: SQL
Return Type: void

staff_lasso_metarecord_copy_count(transcendant integer, unshadow bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

staff_lasso_record_copy_count(transcendant integer, unshadow bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

staff_ou_metarecord_copy_count(transcendant integer, unshadow bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

staff_ou_record_copy_count(transcendant integer, unshadow bigint)

Function Properties

Language: PLPGSQL

Return Type: SET OF record

Schema auditor

Table: acq_invoice_entry_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- PRIMARY KEY,
audit_time : timestamp with time zone -- NOT NULL,
audit_action : text -- NOT NULL,
id : integer -- NOT NULL,
invoice : integer -- NOT NULL,
purchase_order : integer --
lineitem : integer --
inv_item_count : integer -- NOT NULL,
phys_item_count : integer --
note : text --
billed_per_item : boolean --
cost_billed : numeric(8,2) --
actual_cost : numeric(8,2) --
amount_paid : numeric(8,2) --

View: acq_invoice_entry_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : integer --
invoice : integer --
purchase_order : integer --
lineitem : integer --
inv_item_count : integer --
phys_item_count : integer --
note : text --
billed_per_item : boolean --
cost_billed : numeric(8,2) --
actual_cost : numeric(8,2) --
amount_paid : numeric(8,2) --

Table: acq_invoice_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- PRIMARY KEY,
audit_time : timestamp with time zone -- NOT NULL,
audit_action : text -- NOT NULL,
id : integer -- NOT NULL,
receiver : integer -- NOT NULL,
provider : integer -- NOT NULL,
shipper : integer -- NOT NULL,
recv_date : timestamp with time zone -- NOT NULL,
recv_method : text -- NOT NULL,
inv_type : text --
inv_ident : text -- NOT NULL,
payment_auth : text --
payment_method : text --
note : text --
complete : boolean -- NOT NULL,

Table: acq_invoice_item_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- PRIMARY KEY,
audit_time : timestamp with time zone -- NOT NULL,
audit_action : text -- NOT NULL,
id : integer -- NOT NULL,
invoice : integer -- NOT NULL,
purchase_order : integer --
fund_debit : integer --
inv_item_type : text -- NOT NULL,
title : text --
author : text --
note : text --
cost_billed : numeric(8,2) --
actual_cost : numeric(8,2) --
fund : integer --
amount_paid : numeric(8,2) --
po_item : integer --
target : bigint --

View: acq_invoice_item_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : integer --
invoice : integer --

purchase_order : integer --
fund_debit : integer --
inv_item_type : text --
title : text --
author : text --
note : text --
cost_billed : numeric(8,2) --
actual_cost : numeric(8,2) --
fund : integer --
amount_paid : numeric(8,2) --
po_item : integer --
target : bigint --

View: acq_invoice_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : integer --
receiver : integer --
provider : integer --
shipper : integer --
recv_date : timestamp with time zone --
recv_method : text --
inv_type : text --
inv_ident : text --
payment_auth : text --
payment_method : text --
note : text --
complete : boolean --

Table: actor_org_unit_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- `PRIMARY KEY`,
audit_time : timestamp with time zone -- `NOT NULL`,
audit_action : text -- `NOT NULL`,
id : integer -- `NOT NULL`,
parent_ou : integer --
ou_type : integer -- `NOT NULL`,
ill_address : integer --
holds_address : integer --
mailing_address : integer --
billing_address : integer --
shortname : text -- `NOT NULL`,

name : text -- NOT NULL,
email : text --
phone : text --
opac_visible : boolean -- NOT NULL,
fiscal_calendar : integer -- NOT NULL,

View: actor_org_unit_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : integer --
parent_ou : integer --
ou_type : integer --
ill_address : integer --
holds_address : integer --
mailing_address : integer --
billing_address : integer --
shortname : text --
name : text --
email : text --
phone : text --
opac_visible : boolean --
fiscal_calendar : integer --

Table: actor_usr_address_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- PRIMARY KEY,
audit_time : timestamp with time zone -- NOT NULL,
audit_action : text -- NOT NULL,
id : integer -- NOT NULL,
valid : boolean -- NOT NULL,
within_city_limits : boolean -- NOT NULL,
address_type : text -- NOT NULL,
usr : integer -- NOT NULL,
street1 : text -- NOT NULL,
street2 : text --
city : text -- NOT NULL,
county : text --
state : text -- NOT NULL,
country : text -- NOT NULL,
post_code : text -- NOT NULL,
pending : boolean -- NOT NULL,
replaces : integer --

Indexes:

aud_actor_usr_address_hist_id_idx : id

View: actor_usr_address_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : integer --
valid : boolean --
within_city_limits : boolean --
address_type : text --
usr : integer --
street1 : text --
street2 : text --
city : text --
county : text --
state : text --
country : text --
post_code : text --
pending : boolean --
replaces : integer --

Table: actor_usr_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- PRIMARY KEY,
audit_time : timestamp with time zone -- NOT NULL,
audit_action : text -- NOT NULL,
id : integer -- NOT NULL,
card : integer --
profile : integer -- NOT NULL,
usrname : text -- NOT NULL,
email : text --
passwd : text -- NOT NULL,
standing : integer -- NOT NULL,
ident_type : integer -- NOT NULL,
ident_value : text --
ident_type2 : integer --
ident_value2 : text --
net_access_level : integer -- NOT NULL,
photo_url : text --
prefix : text --
first_given_name : text -- NOT NULL,
second_given_name : text --

family_name : text -- NOT NULL,
suffix : text --
alias : text --
day_phone : text --
evening_phone : text --
other_phone : text --
mailing_address : integer --
billing_address : integer --
home_ou : integer -- NOT NULL,
dob : timestamp with time zone --
active : boolean -- NOT NULL,
master_account : boolean -- NOT NULL,
super_user : boolean -- NOT NULL,
barred : boolean -- NOT NULL,
deleted : boolean -- NOT NULL,
juvenile : boolean -- NOT NULL,
usrgroup : integer -- NOT NULL,
claims_returned_count : integer -- NOT NULL,
credit_forward_balance : numeric(6,2) -- NOT NULL,
last_xact_id : text -- NOT NULL,
alert_message : text --
create_date : timestamp with time zone -- NOT NULL,
expire_date : timestamp with time zone -- NOT NULL,
claims_never_checked_out_count : integer -- NOT NULL,

Indexes:

aud_actor_usr_hist_id_idx : id

View: actor_usr_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : integer --
card : integer --
profile : integer --
usrname : text --
email : text --
passwd : text --
standing : integer --
ident_type : integer --
ident_value : text --
ident_type2 : integer --
ident_value2 : text --
net_access_level : integer --
photo_url : text --
prefix : text --

first_given_name : text --
second_given_name : text --
family_name : text --
suffix : text --
alias : text --
day_phone : text --
evening_phone : text --
other_phone : text --
mailing_address : integer --
billing_address : integer --
home_ou : integer --
dob : timestamp with time zone --
active : boolean --
master_account : boolean --
super_user : boolean --
barred : boolean --
deleted : boolean --
juvenile : boolean --
usrgroup : integer --
claims_returned_count : integer --
credit_forward_balance : numeric(6,2) --
last_xact_id : text --
alert_message : text --
create_date : timestamp with time zone --
expire_date : timestamp with time zone --
claims_never_checked_out_count : integer --

Table: asset_call_number_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- `PRIMARY KEY`,
audit_time : timestamp with time zone -- `NOT NULL`,
audit_action : text -- `NOT NULL`,
id : bigint -- `NOT NULL`,
creator : bigint -- `NOT NULL`,
create_date : timestamp with time zone --
editor : bigint -- `NOT NULL`,
edit_date : timestamp with time zone --
record : bigint -- `NOT NULL`,
owning_lib : integer -- `NOT NULL`,
label : text -- `NOT NULL`,
deleted : boolean -- `NOT NULL`,
label_class : bigint -- `NOT NULL`,
label_sortkey : text --

Indexes:

aud_asset_cn_hist_creator_idx : creator
aud_asset_cn_hist_editor_idx : editor

View: asset_call_number_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : bigint --
creator : bigint --
create_date : timestamp with time zone --
editor : bigint --
edit_date : timestamp with time zone --
record : bigint --
owning_lib : integer --
label : text --
deleted : boolean --
label_class : bigint --
label_sortkey : text --

Table: asset_copy_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- PRIMARY KEY,
audit_time : timestamp with time zone -- NOT NULL,
audit_action : text -- NOT NULL,
id : bigint -- NOT NULL,
circ_lib : integer -- NOT NULL,
creator : bigint -- NOT NULL,
call_number : bigint -- NOT NULL,
editor : bigint -- NOT NULL,
create_date : timestamp with time zone --
edit_date : timestamp with time zone --
copy_number : integer --
status : integer -- NOT NULL,
location : integer -- NOT NULL,
loan_duration : integer -- NOT NULL,
fine_level : integer -- NOT NULL,
age_protect : integer --
circulate : boolean -- NOT NULL,
deposit : boolean -- NOT NULL,
ref : boolean -- NOT NULL,
holdable : boolean -- NOT NULL,
deposit_amount : numeric(6,2) -- NOT NULL,
price : numeric(8,2) --
barcode : text -- NOT NULL,
circ_modifier : text --
circ_as_type : text --

dummy_title : text --
dummy_author : text --
alert_message : text --
opac_visible : boolean -- NOT NULL,
deleted : boolean -- NOT NULL,
floating : boolean -- NOT NULL,
dummy_isbn : text --
status_changed_time : timestamp with time zone --
mint_condition : boolean -- NOT NULL,
cost : numeric(8,2) --

Indexes:

aud_asset_cp_hist_creator_idx : creator
aud_asset_cp_hist_editor_idx : editor

View: asset_copy_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : bigint --
circ_lib : integer --
creator : bigint --
call_number : bigint --
editor : bigint --
create_date : timestamp with time zone --
edit_date : timestamp with time zone --
copy_number : integer --
status : integer --
location : integer --
loan_duration : integer --
fine_level : integer --
age_protect : integer --
circulate : boolean --
deposit : boolean --
ref : boolean --
holdable : boolean --
deposit_amount : numeric(6,2) --
price : numeric(8,2) --
barcode : text --
circ_modifier : text --
circ_as_type : text --
dummy_title : text --
dummy_author : text --
alert_message : text --
opac_visible : boolean --
deleted : boolean --

floating : boolean --
dummy_isbn : text --
status_changed_time : timestamp with time zone --
mint_condition : boolean --
cost : numeric(8,2) --

Table: biblio_record_entry_history

Columns:

field name : datatype -- parameters, contraints and notes
audit_id : bigint -- PRIMARY KEY,
audit_time : timestamp with time zone -- NOT NULL,
audit_action : text -- NOT NULL,
id : bigint -- NOT NULL,
creator : integer -- NOT NULL,
editor : integer -- NOT NULL,
source : integer --
quality : integer --
create_date : timestamp with time zone -- NOT NULL,
edit_date : timestamp with time zone -- NOT NULL,
active : boolean -- NOT NULL,
deleted : boolean -- NOT NULL,
fingerprint : text --
tcn_source : text -- NOT NULL,
tcn_value : text -- NOT NULL,
marc : text -- NOT NULL,
last_xact_id : text -- NOT NULL,
owner : integer --
share_depth : integer --

Indexes:

aud_bib_rec_entry_hist_creator_idx : creator
aud_bib_rec_entry_hist_editor_idx : editor

View: biblio_record_entry_lifecycle

Columns:

field name : datatype -- parameters, contraints and notes
?column? : bigint --
audit_time : timestamp with time zone --
audit_action : text --
id : bigint --
creator : integer --
editor : integer --
source : integer --
quality : integer --
create_date : timestamp with time zone --
edit_date : timestamp with time zone --

active : boolean --
deleted : boolean --
fingerprint : text --
tcn_source : text --
tcn_value : text --
marc : text --
last_xact_id : text --
owner : integer --
share_depth : integer --

audit_acq_invoice_entry_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

audit_acq_invoice_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

audit_acq_invoice_item_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

audit_actor_org_unit_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

audit_actor_usr_address_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

audit_actor_usr_func()

Function Properties

Language: PLPGSQL

Return Type: trigger

audit_asset_call_number_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

audit_asset_copy_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

audit_biblio_record_entry_func()

Function Properties

Language: PLPGSQL
Return Type: trigger

create_auditor(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

create_auditor_func(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

create_auditor_history(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

create_auditor_lifecycle(tbl text, sch text)

Function Properties

Language: PLPGSQL

Return Type: boolean

create_auditor_seq(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

create_auditor_update_trigger(tbl text, sch text)

Function Properties

Language: PLPGSQL
Return Type: boolean

Schema authority

Table: bib_linking

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
bib : bigint -- `NOT NULL`, `REFERENCES` biblio.record_entry.
authority : bigint -- `NOT NULL`, `REFERENCES` authority.record_entry.

Indexes:

authority_bl_bib_idx : bib

Table: full_rec

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
record : bigint -- `NOT NULL`,
tag : character(3) -- `NOT NULL`,
ind1 : text --
ind2 : text --
subfield : text --
value : text -- `NOT NULL`,
index_vector : tsvector -- `NOT NULL`,

Indexes:

authority_full_rec_index_vector_idx : index_vector
authority_full_rec_record_idx : record

authority_full_rec_tag_part_idx : "substring"((tag)::text, 2)
authority_full_rec_tag_subfield_idx : tag, subfield
authority_full_rec_value_tpo_index : value text_pattern_ops

Table: rec_descriptor

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
record : bigint --
record_status : text --
char_encoding : text --

Indexes:

authority_rec_descriptor_record_idx : record

Table: record_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
creator : integer -- `NOT NULL`, `DEFAULT 1`,
editor : integer -- `NOT NULL`, `DEFAULT 1`,
create_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
edit_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
active : boolean -- `NOT NULL`, `DEFAULT true`,
deleted : boolean -- `NOT NULL`, `DEFAULT false`,
source : integer --
marc : text -- `NOT NULL`,
last_xact_id : text -- `NOT NULL`,
owner : integer --

Indexes:

authority_record_entry_creator_idx : creator
authority_record_entry_editor_idx : editor
by_heading_and_thesaurus : authority.normalize_heading(marc)) WHERE ((deleted IS FALSE) OR (deleted = false)

Tables referencing authority.bib_linking via Foreign Key Constraints:

authority.bib_linking
authority.record_note
vandelay.authority_match
vandelay.queued_authority_record

Table: record_note

Columns:

field name : datatype -- parameters, contraints and notes

id : bigserial -- PRIMARY KEY,
record : bigint -- NOT NULL, REFERENCES authority.record_entry.
value : text -- NOT NULL,
creator : integer -- NOT NULL, DEFAULT 1,
editor : integer -- NOT NULL, DEFAULT 1,
create_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
edit_date : timestamp with time zone -- NOT NULL, DEFAULT now(),

Indexes:

authority_record_note_creator_idx : creator
authority_record_note_editor_idx : editor
authority_record_note_record_idx : record

View: tracing_links

Columns:

field name : datatype -- parameters, contraints and notes
record : bigint --
main_id : bigint --
main_tag : character(3) --
main_value : text --
relationship : text --
use_restriction : text --
deprecation : text --
display_restriction : text --
link_id : bigint --
link_tag : character(3) --
link_value : text --

flatten_marc(rid bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF full_rec

flatten_marc(text)

Function Properties

Language: PLPERLU
Return Type: SET OF full_rec

generate_overlay_template(bigint)

Function Properties

Language: SQL

Return Type: text

generate_overlay_template(text)

Function Properties

Language: SQL
Return Type: text

generate_overlay_template(text, bigint)

Function Properties

Language: PLPERLU
Return Type: text

indexing_ingest_or_delete()

Function Properties

Language: PLPGSQL
Return Type: trigger

merge_records(source_record bigint, target_record bigint)

Function Properties

Language: PLPGSQL
Return Type: integer

normalize_heading(text)

Extract the authority heading, thesaurus, and NACO-normalized values from an authority record. The primary purpose is to build a unique index to defend against duplicated authority records from the same thesaurus.

Function Properties

Language: PLPERLU
Return Type: text

propagate_changes(aid bigint)

Function Properties

Language: SQL
Return Type: SET OF bigint

propagate_changes(bid bigint, aid bigint)

Function Properties

Language: SQL
Return Type: bigint

reingest_authority_full_rec(auth_id bigint)

Function Properties

Language: PLPGSQL
Return Type: void

reingest_authority_rec_descriptor(auth_id bigint)

Function Properties

Language: PLPGSQL
Return Type: void

Schema biblio

Table: record_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
creator : integer -- `NOT NULL`, `DEFAULT 1`, `REFERENCES` actor.usr.
editor : integer -- `NOT NULL`, `DEFAULT 1`, `REFERENCES` actor.usr.
source : integer --
quality : integer --
create_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
edit_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
active : boolean -- `NOT NULL`, `DEFAULT true`,
deleted : boolean -- `NOT NULL`, `DEFAULT false`,
fingerprint : text --
tcn_source : text -- `NOT NULL`, `DEFAULT 'AUTOGEN'::text`,
tcn_value : text -- `NOT NULL`, `DEFAULT biblio.next_autogen_tcn_value()`,
marc : text -- `NOT NULL`,
last_xact_id : text -- `NOT NULL`,
owner : integer -- `REFERENCES` actor.org_unit.
share_depth : integer --

Indexes:

biblio_record_entry_create_date_idx : create_date

biblio_record_entry_creator_idx : creator
biblio_record_entry_edit_date_idx : edit_date
biblio_record_entry_editor_idx : editor
biblio_record_entry_fp_idx : fingerprint

Tables referencing acq.lineitem via Foreign Key Constraints:

acq.lineitem
asset.call_number
biblio.record_note
container.biblio_record_entry_bucket_item
metabib.identifier_field_entry
metabib.metarecord
metabib.real_full_rec
metabib.series_field_entry
metabib.title_field_entry
serial.subscription
vandelay.queued_bib_record
acq.user_request
authority.bib_linking
booking.resource_type
metabib.author_field_entry
metabib.keyword_field_entry
metabib.metarecord_source_map
metabib.rec_descriptor
metabib.subject_field_entry
serial.record_entry
vandelay.bib_match

Table: record_note

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
record : bigint -- `NOT NULL`, `REFERENCES` biblio.record_entry.
value : text -- `NOT NULL`,
creator : integer -- `NOT NULL`, `DEFAULT 1`, `REFERENCES` actor.usr.
editor : integer -- `NOT NULL`, `DEFAULT 1`, `REFERENCES` actor.usr.
pub : boolean -- `NOT NULL`, `DEFAULT false`,
create_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
edit_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,

Indexes:

biblio_record_note_creator_idx : creator
biblio_record_note_editor_idx : editor
biblio_record_note_record_idx : record

check_marcxml_well_formed()

Function Properties

Language: PLPGSQL
Return Type: trigger

extract_fingerprint(marc text)

Function Properties

Language: PLPGSQL

Return Type: text

extract_located_uris(editor_id bigint, marcxml text, bib_id integer)

Function Properties

Language: PLPGSQL
Return Type: void

extract_metabib_field_entry(bigint)

Function Properties

Language: SQL
Return Type: SET OF field_entry_template

extract_metabib_field_entry(default_joiner bigint, rid text)

Function Properties

Language: PLPGSQL
Return Type: SET OF field_entry_template

extract_quality(best_type text, best_lang text, marc text)

Function Properties

Language: PLPGSQL
Return Type: integer

fingerprint_trigger()

Function Properties

Language: PLPGSQL
Return Type: trigger

flatten_marc(rid bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF full_rec

flatten_marc(text)

Function Properties

Language: PLPERLU
Return Type: SET OF full_rec

indexing_ingest_or_delete()

Function Properties

Language: PLPGSQL
Return Type: trigger

map_authority_linking(marc bigint, bibid text)

Function Properties

Language: SQL
Return Type: bigint

marc21_extract_fixed_field(ff bigint, rid text)

Function Properties

Language: PLPGSQL
Return Type: text

marc21_physical_characteristics(rid bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF marc21_physical_characteristics

marc21_record_type(rid bigint)

Function Properties

Language: PLPGSQL
Return Type: marc21_rec_type_map

next_autogen_tcn_value()

Function Properties

Language: PLPGSQL

Return Type: text

Schema booking

Table: reservation

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`, `DEFAULT nextval('money.billable_xact_id_seq'::regclass)`,
usr : integer -- `NOT NULL`, `REFERENCES` actor.usr.
xact_start : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
xact_finish : timestamp with time zone --
unrecovered : boolean --
request_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
start_time : timestamp with time zone --
end_time : timestamp with time zone --
capture_time : timestamp with time zone --
cancel_time : timestamp with time zone --
pickup_time : timestamp with time zone --
return_time : timestamp with time zone --
booking_interval : interval --
fine_interval : interval --
fine_amount : numeric(8,2) --
max_fine : numeric(8,2) --
target_resource_type : integer -- `NOT NULL`, `REFERENCES` booking.resource_type.
target_resource : integer -- `REFERENCES` booking.resource.
current_resource : integer -- `REFERENCES` booking.resource.
request_lib : integer -- `NOT NULL`, `REFERENCES` actor.org_unit.
pickup_lib : integer -- `REFERENCES` actor.org_unit.
capture_staff : integer -- `REFERENCES` actor.usr.

Tables referencing action.reservation_transit_copy via Foreign Key Constraints:

action.reservation_transit_copy

booking.reservation_attr_value_map

Table: reservation_attr_value_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
reservation : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` booking.reservation.
attr_value : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` booking.resource_attr_value.

Table: resource

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
type : integer -- NOT NULL, REFERENCES booking.resource_type.
overbook : boolean -- NOT NULL, DEFAULT false,
barcode : text -- UNIQUE#1, NOT NULL,
deposit : boolean -- NOT NULL, DEFAULT false,
deposit_amount : numeric(8,2) -- NOT NULL, DEFAULT 0.00,
user_fee : numeric(8,2) -- NOT NULL, DEFAULT 0.00,

Tables referencing action.reservation_transit_copy via Foreign Key Constraints:

action.reservation_transit_copy
booking.reservation
booking.resource_attr_map

Table: resource_attr

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owner : integer -- NOT NULL, REFERENCES actor.org_unit.
name : text -- UNIQUE#1, NOT NULL,
resource_type : integer -- UNIQUE#1, NOT NULL, REFERENCES booking.resource_type.
required : boolean -- NOT NULL, DEFAULT false,

Tables referencing booking.resource_attr_map via Foreign Key Constraints:

booking.resource_attr_map
booking.resource_attr_value

Table: resource_attr_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
resource : integer -- UNIQUE#1, NOT NULL, REFERENCES booking.resource.
resource_attr : integer -- UNIQUE#1, NOT NULL, REFERENCES booking.resource_attr.
value : integer -- NOT NULL, REFERENCES booking.resource_attr_value.

Table: resource_attr_value

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
attr : integer -- UNIQUE#1, NOT NULL, REFERENCES booking.resource_attr.
valid_value : text -- UNIQUE#1, NOT NULL,

Tables referencing booking.reservation_attr_value_map via Foreign Key Constraints:

booking.reservation_attr_value_map booking.resource_attr_map

Table: resource_type

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE#1, NOT NULL,
elbow_room : interval --
fine_interval : interval --
fine_amount : numeric(8,2) -- NOT NULL,
max_fine : numeric(8,2) --
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
catalog_item : boolean -- NOT NULL, DEFAULT false,
transferable : boolean -- NOT NULL, DEFAULT false,
record : bigint -- UNIQUE#1, REFERENCES biblio.record_entry.

Tables referencing booking.reservation via Foreign Key Constraints:

booking.reservation booking.resource
booking.resource_attr

Schema config

The config schema holds static configuration data for the Evergreen installation.

Table: audience_map

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
value : text -- NOT NULL,
description : text --

Table: bib_level_map

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
value : text -- NOT NULL,

Table: bib_source

This is table is used to set up the relative "quality" of each MARC source, such as OCLC. Also identifies "transcendant" sources, i.e., sources of bib records that should display in the OPAC even if no copies or located URIs are attached.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
quality : integer --
source : text -- UNIQUE, NOT NULL,
transcendant : boolean -- NOT NULL, DEFAULT false,

Constraints:

bib_source_quality_check : CHECK (((quality >= 0) AND (quality <= 100)))

Tables referencing vandelay.queued_bib_record via Foreign Key Constraints:

vandelay.queued_bib_record

Table: biblio_fingerprint

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- NOT NULL,
xpath : text -- NOT NULL,
first_word : boolean -- NOT NULL, DEFAULT false,
format : text -- NOT NULL, DEFAULT 'marcxml'::text,

Table: billing_type

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE#1, NOT NULL,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
default_price : numeric(6,2) --

Tables referencing money.billing via Foreign Key Constraints:

money.billing

Table: circ_matrix_circ_mod_test

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
matchpoint : integer -- NOT NULL, REFERENCES config.circ_matrix_matchpoint.
items_out : integer -- NOT NULL,

Tables referencing config.circ_matrix_circ_mod_test_map via Foreign Key Constraints:

config.circ_matrix_circ_mod_test_map

Table: circ_matrix_circ_mod_test_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
circ_mod_test : integer -- UNIQUE#1, NOT NULL, REFERENCES config.circ_matrix_circ_mod_test.
circ_mod : text -- UNIQUE#1, NOT NULL, REFERENCES config.circ_modifier.

Table: circ_matrix_matchpoint

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
active : boolean -- NOT NULL, DEFAULT true,
org_unit : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
grp : integer -- UNIQUE#1, NOT NULL, REFERENCES permission.grp_tree.
circ_modifier : text -- UNIQUE#1, REFERENCES config.circ_modifier.
marc_type : text -- UNIQUE#1, REFERENCES config.item_type_map.
marc_form : text -- UNIQUE#1, REFERENCES config.item_form_map.
marc_vr_format : text -- UNIQUE#1, REFERENCES config.videorecording_format_map.
copy_circ_lib : integer -- UNIQUE#1, REFERENCES actor.org_unit.
copy_owning_lib : integer -- UNIQUE#1, REFERENCES actor.org_unit.
ref_flag : boolean -- UNIQUE#1,
juvenile_flag : boolean -- UNIQUE#1,
is_renewal : boolean -- UNIQUE#1,
usr_age_lower_bound : interval -- UNIQUE#1,
usr_age_upper_bound : interval -- UNIQUE#1,
circulate : boolean -- NOT NULL, DEFAULT true,
duration_rule : integer -- NOT NULL, REFERENCES config.rule_circ_duration.
recurring_fine_rule : integer -- NOT NULL, REFERENCES config.rule_recurring_fine.
max_fine_rule : integer -- NOT NULL, REFERENCES config.rule_max_fine.
hard_due_date : integer -- REFERENCES config.hard_due_date.
script_test : text --
total_copy_hold_ratio : double precision --
available_copy_hold_ratio : double precision --

Tables referencing config.circ_matrix_circ_mod_test via Foreign Key Constraints:

config.circ_matrix_circ_mod_test

Table: circ_modifier

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
name : text -- UNIQUE, NOT NULL,

description : text -- NOT NULL,
sip2_media_type : text -- NOT NULL,
magnetic_media : boolean -- NOT NULL, DEFAULT true,
avg_wait_time : interval --

Tables referencing acq.lineitem_detail via Foreign Key Constraints:

acq.lineitem_detail
asset.copy
config.circ_matrix_circ_mod_test_map
config.circ_matrix_matchpoint
config.hold_matrix_matchpoint

Table: copy_status

Copy Statuses The available copy statuses, and whether a copy in that status is available for hold request capture. 0 (zero) is the only special number in this set, meaning that the item is available for immediate checkout, and is counted as available in the OPAC. Statuses with an ID below 100 are not removable, and have special meaning in the code. Do not change them except to translate the textual name. You may add and remove statuses above 100, and these can be used to remove items from normal circulation without affecting the rest of the copy's values or its location.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE, NOT NULL,
holdable : boolean -- NOT NULL, DEFAULT false,
opac_visible : boolean -- NOT NULL, DEFAULT false,

Tables referencing action.transit_copy via Foreign Key Constraints:

action.transit_copy
asset.copy
asset.copy_template

Table: global_flag

Columns:

field name : datatype -- parameters, contraints and notes
name : text -- PRIMARY KEY,
value : text --
enabled : boolean -- NOT NULL, DEFAULT false,
label : text -- NOT NULL,

Table: hard_due_date

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE, NOT NULL,
ceiling_date : timestamp with time zone -- NOT NULL,
forceto : boolean -- NOT NULL,

owner : integer -- NOT NULL,

Constraints:

hard_due_date_name_check : CHECK ((name ~ '^\\w+$'::text))

Tables referencing config.circ_matrix_matchpoint via Foreign Key Constraints:

config.circ_matrix_matchpoint config.hard_due_date_values

Table: hard_due_date_values

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
hard_due_date : integer -- NOT NULL, REFERENCES config.hard_due_date.
ceiling_date : timestamp with time zone -- NOT NULL,
active_date : timestamp with time zone -- NOT NULL,

Table: hold_matrix_matchpoint

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
active : boolean -- NOT NULL, DEFAULT true,
user_home_ou : integer -- UNIQUE#1, REFERENCES actor.org_unit.
request_ou : integer -- UNIQUE#1, REFERENCES actor.org_unit.
pickup_ou : integer -- UNIQUE#1, REFERENCES actor.org_unit.
item_owning_ou : integer -- UNIQUE#1, REFERENCES actor.org_unit.
item_circ_ou : integer -- UNIQUE#1, REFERENCES actor.org_unit.
usr_grp : integer -- UNIQUE#1, REFERENCES permission.grp_tree.
requestor_grp : integer -- UNIQUE#1, NOT NULL, REFERENCES permission.grp_tree.
circ_modifier : text -- UNIQUE#1, REFERENCES config.circ_modifier.
marc_type : text -- UNIQUE#1, REFERENCES config.item_type_map.
marc_form : text -- UNIQUE#1, REFERENCES config.item_form_map.
marc_vr_format : text -- UNIQUE#1, REFERENCES config.videorecording_format_map.
juvenile_flag : boolean -- UNIQUE#1,
ref_flag : boolean -- UNIQUE#1,
holdable : boolean -- NOT NULL, DEFAULT true,
distance_is_from_owner : boolean -- NOT NULL, DEFAULT false,
transit_range : integer -- REFERENCES actor.org_unit_type.
max_holds : integer --
include_frozen_holds : boolean -- NOT NULL, DEFAULT true,
stop_blocked_user : boolean -- NOT NULL, DEFAULT false,
age_hold_protect_rule : integer -- REFERENCES config.rule_age_hold_protect.

Table: i18n_core

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
fq_field : text -- NOT NULL,
identity_value : text -- NOT NULL,
translation : text -- NOT NULL, REFERENCES config.i18n_locale.
string : text -- NOT NULL,

Table: i18n_locale

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
marc_code : text -- NOT NULL, REFERENCES config.language_map.
name : text -- UNIQUE, NOT NULL,
description : text --

Tables referencing config.i18n_core via Foreign Key Constraints:

config.i18n_core

Table: identification_type

Types of valid patron identification. Each patron must display at least one valid form of identification in order to get a library card. This table lists those forms.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE, NOT NULL,

Tables referencing actor.usr via Foreign Key Constraints:

actor.usr

Table: idl_field_doc

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
fm_class : text -- NOT NULL,
field : text -- NOT NULL,
owner : integer -- NOT NULL, REFERENCES actor.org_unit.
string : text -- NOT NULL,

Table: index_normalizer

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE, NOT NULL,
description : text --
func : text -- NOT NULL,
param_count : integer -- NOT NULL,

Tables referencing config.metabib_field_index_norm_map via Foreign Key Constraints:

config.metabib_field_index_norm_map

Table: internal_flag

Columns:

field name : datatype -- parameters, contraints and notes
name : text -- PRIMARY KEY,
value : text --
enabled : boolean -- NOT NULL, DEFAULT false,

Table: item_form_map

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
value : text -- NOT NULL,

Tables referencing config.circ_matrix_matchpoint via Foreign Key Constraints:

config.circ_matrix_matchpoint config.hold_matrix_matchpoint

Table: item_type_map

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
value : text -- NOT NULL,

Tables referencing config.circ_matrix_matchpoint via Foreign Key Constraints:

config.circ_matrix_matchpoint config.hold_matrix_matchpoint

Table: language_map

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,

value : text -- NOT NULL,

Tables referencing config.i18n_locale via Foreign Key Constraints:

config.i18n_locale

Table: lit_form_map

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
value : text -- NOT NULL,
description : text --

Table: marc21_ff_pos_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
fixed_field : text -- NOT NULL,
tag : text -- NOT NULL,
rec_type : text -- NOT NULL,
start_pos : integer -- NOT NULL,
length : integer -- NOT NULL,
default_val : text -- NOT NULL, DEFAULT ' '::text,

Table: marc21_physical_characteristic_subfield_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
ptype_key : text -- NOT NULL, REFERENCES config.marc21_physical_characteristic_type_map.
subfield : text -- NOT NULL,
start_pos : integer -- NOT NULL,
length : integer -- NOT NULL,
label : text -- NOT NULL,

Tables referencing config.marc21_physical_characteristic_value_map via Foreign Key Constraints:

config.marc21_physical_characteristic_value_map

Table: marc21_physical_characteristic_type_map

Columns:

field name : datatype -- parameters, contraints and notes
ptype_key : text -- PRIMARY KEY,

label : text -- NOT NULL,

Tables referencing config.marc21_physical_characteristic_subfield_map via Foreign Key Constraints:

config.marc21_physical_characteristic_subfield_map

Table: marc21_physical_characteristic_value_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
value : text -- NOT NULL,
ptype_subfield : integer -- NOT NULL, REFERENCES config.marc21_physical_characteristic_subfield_map.
label : text -- NOT NULL,

Table: marc21_rec_type_map

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
type_val : text -- NOT NULL,
blvl_val : text -- NOT NULL,

Table: metabib_class

Columns:

field name : datatype -- parameters, contraints and notes
name : text -- PRIMARY KEY,
label : text -- UNIQUE, NOT NULL,

Tables referencing config.metabib_field via Foreign Key Constraints:

config.metabib_field config.metabib_search_alias

Table: metabib_field

XPath used for record indexing ingest This table contains the XPath used to chop up MODS into its indexable parts. Each XPath entry is named and assigned to a "class" of either title, subject, author, keyword, series or identifier.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
field_class : text -- NOT NULL, REFERENCES config.metabib_class.
name : text -- NOT NULL,
label : text -- NOT NULL,
xpath : text -- NOT NULL,
weight : integer -- NOT NULL, DEFAULT 1,

format : text -- NOT NULL, DEFAULT 'mods33'::text, REFERENCES config.xml_transform.
search_field : boolean -- NOT NULL, DEFAULT true,
facet_field : boolean -- NOT NULL, DEFAULT false,
facet_xpath : text --

Tables referencing config.metabib_field_index_norm_map via Foreign Key Constraints:

config.metabib_field_index_norm_map
config.metabib_search_alias
metabib.author_field_entry
metabib.identifier_field_entry
metabib.keyword_field_entry
metabib.series_field_entry
metabib.subject_field_entry
metabib.title_field_entry
search.relevance_adjustment

Table: metabib_field_index_norm_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
field : integer -- NOT NULL, REFERENCES config.metabib_field.
norm : integer -- NOT NULL, REFERENCES config.index_normalizer.
params : text --
pos : integer -- NOT NULL,

Table: metabib_search_alias

Columns:

field name : datatype -- parameters, contraints and notes
alias : text -- PRIMARY KEY,
field_class : text -- NOT NULL, REFERENCES config.metabib_class.
field : integer -- REFERENCES config.metabib_field.

Table: net_access_level

Patron Network Access level This will be used to inform the in-library firewall of how much internet access the using patron should be allowed.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE, NOT NULL,

Tables referencing actor.usr via Foreign Key Constraints:

actor.usr

Table: non_cataloged_type

Types of valid non-cataloged items.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owning_lib : integer -- UNIQUE#1, NOT NULL,
name : text -- UNIQUE#1, NOT NULL,
circ_duration : interval -- NOT NULL, DEFAULT '14 days'::interval,
in_house : boolean -- NOT NULL, DEFAULT false,

Tables referencing action.non_cat_in_house_use via Foreign Key Constraints:

action.non_cat_in_house_use action.non_cataloged_circulation

Table: org_unit_setting_type

Columns:

field name : datatype -- parameters, contraints and notes
name : text -- PRIMARY KEY,
label : text -- UNIQUE, NOT NULL,
grp : text -- REFERENCES config.settings_group.
description : text --
datatype : text -- NOT NULL, DEFAULT 'string'::text,
fm_class : text --
view_perm : integer -- REFERENCES permission.perm_list.
update_perm : integer -- REFERENCES permission.perm_list.

Constraints:

coust_no_empty_link : CHECK ((((datatype = 'link'::text) AND (fm_class IS NOT NULL)) OR ((datatype <> 'link'::text) AND (fm_class IS NULL))))
coust_valid_datatype : CHECK ((datatype = ANY (ARRAY['bool'::text, 'integer'::text, 'float'::text, 'currency'::text, 'interval'::text, 'date'::text, 'string'::text, 'object'::text, 'array'::text, 'link'::text])))

Tables referencing actor.org_unit_setting via Foreign Key Constraints:

actor.org_unit_setting

Table: remote_account

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
label : text -- NOT NULL,
host : text -- NOT NULL,
username : text --
password : text --
account : text --
path : text --
owner : integer -- NOT NULL, REFERENCES actor.org_unit.

last_activity : timestamp with time zone --

Table: rule_age_hold_protect

Hold Item Age Protection rules A hold request can only capture new(ish) items when they are within a particular proximity of the pickup_lib of the request. The proximity ('prox' column) is calculated by counting the number of tree edges between the pickup_lib and either the owning_lib or circ_lib of the copy that could fulfill the hold, as determined by the distance_is_from_owner value of the hold matrix rule controlling the hold request.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
name : text -- `UNIQUE`, `NOT NULL`,
age : interval -- `NOT NULL`,
prox : integer -- `NOT NULL`,

Constraints:

rule_age_hold_protect_name_check : CHECK ((name ~ '^\\w+$'::text))

Tables referencing config.hold_matrix_matchpoint via Foreign Key Constraints:

config.hold_matrix_matchpoint

Table: rule_circ_duration

Circulation Duration rules Each circulation is given a duration based on one of these rules.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
name : text -- `UNIQUE`, `NOT NULL`,
extended : interval -- `NOT NULL`,
normal : interval -- `NOT NULL`,
shrt : interval -- `NOT NULL`,
max_renewals : integer -- `NOT NULL`,

Constraints:

rule_circ_duration_name_check : CHECK ((name ~ '^\\w+$'::text))

Tables referencing config.circ_matrix_matchpoint via Foreign Key Constraints:

config.circ_matrix_matchpoint

Table: rule_max_fine

Circulation Max Fine rules Each circulation is given a maximum fine based on one of these rules.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
name : text -- `UNIQUE`, `NOT NULL`,
amount : numeric(6,2) -- `NOT NULL`,
is_percent : boolean -- `NOT NULL`, `DEFAULT false`,

Constraints:

rule_max_fine_name_check : CHECK ((name ~ '^\\w+$'::text))

Tables referencing config.circ_matrix_matchpoint via Foreign Key Constraints:

config.circ_matrix_matchpoint

Table: rule_recurring_fine

Circulation Recurring Fine rules Each circulation is given a recurring fine amount based on one of these rules. Note that it is recommended to run the fine generator (from cron) at least as frequently as the lowest recurrence interval used by your circulation rules so that accrued fines will be up to date.

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
name : text -- `UNIQUE`, `NOT NULL`,
high : numeric(6,2) -- `NOT NULL`,
normal : numeric(6,2) -- `NOT NULL`,
low : numeric(6,2) -- `NOT NULL`,
recurrence_interval : interval -- `NOT NULL`, `DEFAULT '1 day'::interval`,

Constraints:

rule_recurring_fine_name_check : CHECK ((name ~ '^\\w+$'::text))

Tables referencing config.circ_matrix_matchpoint via Foreign Key Constraints:

config.circ_matrix_matchpoint

Table: settings_group

Columns:

field name : datatype -- parameters, contraints and notes
name : text -- `PRIMARY KEY`,
label : text -- `UNIQUE`, `NOT NULL`,

Tables referencing config.org_unit_setting_type via Foreign Key Constraints:

config.org_unit_setting_type

config.usr_setting_type

Table: standing

Patron Standings This table contains the values that can be applied to a patron by a staff member. These values should not be changed, other than for translation, as the ID column is currently a "magic number" in the source. :(

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
value : text -- `UNIQUE`, `NOT NULL`,

Tables referencing actor.usr via Foreign Key Constraints:

actor.usr

Table: standing_penalty

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
name : text -- `UNIQUE`, `NOT NULL`,
label : text -- `NOT NULL`,
block_list : text --
org_depth : integer --

Tables referencing actor.usr_standing_penalty via Foreign Key Constraints:

actor.usr_standing_penalty

permission.grp_penalty_threshold

Table: upgrade_log

Columns:

field name : datatype -- parameters, contraints and notes
version : text -- `PRIMARY KEY`,
install_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,

Table: usr_setting_type

Columns:

field name : datatype -- parameters, contraints and notes
name : text -- `PRIMARY KEY`,
opac_visible : boolean -- `NOT NULL`, `DEFAULT false`,
label : text -- `UNIQUE`, `NOT NULL`,
description : text --
grp : text -- `REFERENCES` config.settings_group.
datatype : text -- `NOT NULL`, `DEFAULT 'string'::text`,

fm_class : text --

Constraints:

coust_no_empty_link : CHECK ((((datatype = 'link'::text) AND (fm_class IS NOT NULL)) OR ((datatype <> 'link'::text) AND (fm_class IS NULL))))
coust_valid_datatype : CHECK ((datatype = ANY (ARRAY['bool'::text, 'integer'::text, 'float'::text, 'currency'::text, 'interval'::text, 'date'::text, 'string'::text, 'object'::text, 'array'::text, 'link'::text])))

Tables referencing action_trigger.event_definition via Foreign Key Constraints:

action_trigger.event_definition actor.usr_setting

Table: videorecording_format_map

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- `PRIMARY KEY`,
value : text -- `NOT NULL`,

Tables referencing config.circ_matrix_matchpoint via Foreign Key Constraints:

config.circ_matrix_matchpoint config.hold_matrix_matchpoint

Table: xml_transform

Columns:

field name : datatype -- parameters, contraints and notes
name : text -- `PRIMARY KEY`,
namespace_uri : text -- `NOT NULL`,
prefix : text -- `NOT NULL`,
xslt : text -- `NOT NULL`,

Tables referencing config.metabib_field via Foreign Key Constraints:

config.metabib_field

Table: z3950_attr

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
source : text -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` config.z3950_source.
name : text -- `NOT NULL`,
label : text -- `NOT NULL`,
code : integer -- `UNIQUE#1`, `NOT NULL`,
format : integer -- `UNIQUE#1`, `NOT NULL`,
truncation : integer -- `NOT NULL`,

Table: z3950_source

Z39.50 Sources Each row in this table represents a database searchable via Z39.50.

Columns:

field name : datatype -- parameters, contraints and notes
name : text -- `PRIMARY KEY`,
label : text -- `UNIQUE`, `NOT NULL`,
host : text -- `NOT NULL`,
port : integer -- `NOT NULL`,
db : text -- `NOT NULL`,
record_format : text -- `NOT NULL`, `DEFAULT 'FI'::text`, Z39.50 element set.
transmission_format : text -- `NOT NULL`, `DEFAULT 'usmarc'::text`, Z39.50 preferred record syntax..
auth : boolean -- `NOT NULL`, `DEFAULT true`,

Tables referencing config.z3950_attr via Foreign Key Constraints:

config.z3950_attr

interval_to_seconds(interval_string text)

Function Properties

Language: PLPGSQL
Return Type: integer

interval_to_seconds(interval_val interval)

Function Properties

Language: PLPGSQL
Return Type: integer

update_hard_due_dates()

Function Properties

Language: PLPGSQL
Return Type: integer

Schema container

Table: biblio_record_entry_bucket

Columns:

field name : datatype -- parameters, contraints and notes

id : serial -- PRIMARY KEY,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.usr.
name : text -- UNIQUE#1, NOT NULL,
btype : text -- UNIQUE#1, NOT NULL, DEFAULT 'misc'::text, REFERENCES container.biblio_record_entry_bucket_type.
pub : boolean -- NOT NULL, DEFAULT false,
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Tables referencing container.biblio_record_entry_bucket_item via Foreign Key Constraints:

container.biblio_record_entry_bucket_item container.biblio_record_entry_bucket_note

Table: biblio_record_entry_bucket_item

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
bucket : integer -- NOT NULL, REFERENCES container.biblio_record_entry_bucket.
target_biblio_record_entry : bigint -- NOT NULL, REFERENCES biblio.record_entry.
pos : integer --
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Tables referencing container.biblio_record_entry_bucket_item_note via Foreign Key Constraints:

container.biblio_record_entry_bucket_item_note

Table: biblio_record_entry_bucket_item_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
item : integer -- NOT NULL, REFERENCES container.biblio_record_entry_bucket_item.
note : text -- NOT NULL,

Table: biblio_record_entry_bucket_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
bucket : integer -- NOT NULL, REFERENCES container.biblio_record_entry_bucket.
note : text -- NOT NULL,

Table: biblio_record_entry_bucket_type

Columns:

field name : datatype -- parameters, contraints and notes

code : text -- PRIMARY KEY,
label : text -- UNIQUE, NOT NULL,

Tables referencing container.biblio_record_entry_bucket via Foreign Key Constraints:

container.biblio_record_entry_bucket

Table: call_number_bucket

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.usr.
name : text -- UNIQUE#1, NOT NULL,
btype : text -- UNIQUE#1, NOT NULL, DEFAULT 'misc'::text, REFERENCES container.call_number_bucket_type.
pub : boolean -- NOT NULL, DEFAULT false,
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Tables referencing container.call_number_bucket_item via Foreign Key Constraints:

container.call_number_bucket_item container.call_number_bucket_note

Table: call_number_bucket_item

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
bucket : integer -- NOT NULL, REFERENCES container.call_number_bucket.
target_call_number : integer -- NOT NULL, REFERENCES asset.call_number.
pos : integer --
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Tables referencing container.call_number_bucket_item_note via Foreign Key Constraints:

container.call_number_bucket_item_note

Table: call_number_bucket_item_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
item : integer -- NOT NULL, REFERENCES container.call_number_bucket_item.
note : text -- NOT NULL,

Table: call_number_bucket_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
bucket : integer -- NOT NULL, REFERENCES container.call_number_bucket.
note : text -- NOT NULL,

Table: call_number_bucket_type

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
label : text -- UNIQUE, NOT NULL,

Tables referencing container.call_number_bucket via Foreign Key Constraints:

container.call_number_bucket

Table: copy_bucket

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.usr.
name : text -- UNIQUE#1, NOT NULL,
btype : text -- UNIQUE#1, NOT NULL, DEFAULT 'misc'::text, REFERENCES container.copy_bucket_type.
pub : boolean -- NOT NULL, DEFAULT false,
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Tables referencing container.copy_bucket_item via Foreign Key Constraints:

container.copy_bucket_item

container.copy_bucket_note

Table: copy_bucket_item

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
bucket : integer -- NOT NULL, REFERENCES container.copy_bucket.
target_copy : integer -- NOT NULL, REFERENCES asset.copy.
pos : integer --
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Indexes:

copy_bucket_item_bucket_idx : bucket

Tables referencing container.copy_bucket_item_note via Foreign Key Constraints:

container.copy_bucket_item_note

Table: copy_bucket_item_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
item : integer -- `NOT NULL`, `REFERENCES` container.copy_bucket_item.
note : text -- `NOT NULL`,

Table: copy_bucket_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
bucket : integer -- `NOT NULL`, `REFERENCES` container.copy_bucket.
note : text -- `NOT NULL`,

Table: copy_bucket_type

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- `PRIMARY KEY`,
label : text -- `UNIQUE`, `NOT NULL`,

Tables referencing container.copy_bucket via Foreign Key Constraints:

container.copy_bucket

Table: user_bucket

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
owner : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.usr.
name : text -- `UNIQUE#1`, `NOT NULL`,
btype : text -- `UNIQUE#1`, `NOT NULL`, `DEFAULT 'misc'::text`, `REFERENCES` container.user_bucket_type.
pub : boolean -- `NOT NULL`, `DEFAULT false`,
create_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,

Tables referencing container.user_bucket_item via Foreign Key Constraints:

container.user_bucket_item

container.user_bucket_note

Table: user_bucket_item

Columns:

field name : datatype -- parameters, contraints and notes

id : serial -- PRIMARY KEY,
bucket : integer -- NOT NULL, REFERENCES container.user_bucket.
target_user : integer -- NOT NULL, REFERENCES actor.usr.
pos : integer --
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),

Indexes:

user_bucket_item_target_user_idx : target_user

Tables referencing container.user_bucket_item_note via Foreign Key Constraints:

container.user_bucket_item_note

Table: user_bucket_item_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
item : integer -- NOT NULL, REFERENCES container.user_bucket_item.
note : text -- NOT NULL,

Table: user_bucket_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
bucket : integer -- NOT NULL, REFERENCES container.user_bucket.
note : text -- NOT NULL,

Table: user_bucket_type

Columns:

field name : datatype -- parameters, contraints and notes
code : text -- PRIMARY KEY,
label : text -- UNIQUE, NOT NULL,

Tables referencing container.user_bucket via Foreign Key Constraints:

container.user_bucket

clear_all_expired_circ_history_items()

Delete expired circulation bucket items for all users that have a setting for patron.max_reading_list_interval.

Function Properties

Language: PLPGSQL

Return Type: void

clear_expired_circ_history_items(ac_usr integer)

Delete old circulation bucket items for a specified user. "Old" means older than the interval specified by a user-level setting, if it is so specified.

Function Properties

Language: PLPGSQL
Return Type: void

Schema evergreen

array_accum(anyelement)

Function Properties

Language: INTERNAL
Return Type: anyarray

change_db_setting(settings text, setting_name text[])

Function Properties

Language: PLPGSQL
Return Type: void

extract_marc_field(text, bigint, text)

Function Properties

Language: SQL
Return Type: text

extract_marc_field(text, bigint, text, text)

Function Properties

Language: PLPGSQL
Return Type: text

facet_force_nfc()

Function Properties

Language: PLPGSQL

Return Type: trigger

fake_fkey_tgr()

Function Properties

Language: PLPGSQL
Return Type: trigger

force_unicode_normal_form(form text, string text)

Function Properties

Language: PLPERLU
Return Type: text

is_json(text)

Function Properties

Language: PLPERLU
Return Type: boolean

lowercase(text)

Function Properties

Language: PLPERLU
Return Type: text

maintain_901()

Function Properties

Language: PLPGSQL
Return Type: trigger

maintain_control_numbers()

Function Properties

Language: PLPERLU
Return Type: trigger

oils_i18n_code_tracking()

Function Properties

Language: PLPGSQL

Return Type: trigger

oils_i18n_gettext(integer, text, text, text)

Function Properties

Language: SQL
Return Type: text

oils_i18n_gettext(text, text, text, text)

Function Properties

Language: SQL
Return Type: text

oils_i18n_id_tracking()

Function Properties

Language: PLPGSQL
Return Type: trigger

oils_i18n_update_apply(hint text, new_ident text, old_ident text)

Function Properties

Language: PLPGSQL
Return Type: void

oils_i18n_xlate(raw_locale text, keyvalue text, identcol text, keycol text, keyclass text, keytable text)

Function Properties

Language: PLPGSQL
Return Type: text

oils_json_to_text(text)

Function Properties

Language: PLPERLU
Return Type: text

oils_text_as_bytea(text)

Function Properties

Language: SQL
Return Type: bytea

oils_tsearch2()

Function Properties

Language: PLPGSQL
Return Type: trigger

oils_xpath(text, text)

Function Properties

Language: SQL
Return Type: text[]

oils_xpath(text, text, anyarray)

Function Properties

Language: SQL
Return Type: text[]

oils_xpath_string(text, text)

Function Properties

Language: SQL
Return Type: text

oils_xpath_string(text, text, anyarray)

Function Properties

Language: SQL
Return Type: text

oils_xpath_string(text, text, text)

Function Properties

Language: SQL

Return Type: text

oils_xpath_string(text, text, text, anyarray)

Function Properties

Language: SQL
Return Type: text

oils_xpath_table(criteria text, xpaths text, relation_name text, document_field text, key text)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

oils_xslt_process(text, text)

Function Properties

Language: PLPERLU
Return Type: text

tableoid2name(oid)

Function Properties

Language: PLPGSQL
Return Type: text

tsvector_concat(public.tsvector, public.tsvector)

Function Properties

Language: SQL
Return Type: tsvector

xml_escape(str text)

Function Properties

Language: SQL
Return Type: text

Schema extend_reporter

View: full_circ_count

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
circ_count : bigint --

View: global_bibs_by_holding_update

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
holding_update : timestamp with time zone --
update_type : text --

Table: legacy_circ_count

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`,
circ_count : integer -- `NOT NULL`,

Schema metabib

Table: author_field_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
source : bigint -- `NOT NULL`, `REFERENCES` biblio.record_entry.
field : integer -- `NOT NULL`, `REFERENCES` config.metabib_field.
value : text -- `NOT NULL`,
index_vector : tsvector -- `NOT NULL`,

Indexes:

metabib_author_field_entry_index_vector_idx : index_vector
metabib_author_field_entry_source_idx : source
metabib_author_field_entry_value_idx : "substring"(value, 1, 1024)) WHERE ((index_vector)::pg_catalog.tsvector = (('::pg_catalog.tsvector)::tsvector)::pg_catalog.tsvector

Table: facet_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
source : bigint -- NOT NULL,
field : integer -- NOT NULL,
value : text -- NOT NULL,

Indexes:

metabib_facet_entry_field_idx : field
metabib_facet_entry_source_idx : source
metabib_facet_entry_value_idx : "substring"(value, 1, 1024)

View: full_rec

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
record : bigint --
tag : character(3) --
ind1 : text --
ind2 : text --
subfield : text --
value : text --
index_vector : tsvector --

Table: identifier_field_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
source : bigint -- NOT NULL, REFERENCES biblio.record_entry.
field : integer -- NOT NULL, REFERENCES config.metabib_field.
value : text -- NOT NULL,
index_vector : tsvector -- NOT NULL,

Indexes:

metabib_identifier_field_entry_index_vector_idx : index_vector
metabib_identifier_field_entry_source_idx : source
metabib_identifier_field_entry_value_idx : "substring"(value, 1, 1024)) WHERE ((index_vector)::pg_catalog.tsvector = (''::pg_catalog.tsvector)::tsvector)::pg_catalog.tsvector

Table: keyword_field_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,

source : bigint -- NOT NULL, REFERENCES biblio.record_entry.
field : integer -- NOT NULL, REFERENCES config.metabib_field.
value : text -- NOT NULL,
index_vector : tsvector -- NOT NULL,

Indexes:

metabib_keyword_field_entry_index_vector_idx : index_vector
metabib_keyword_field_entry_source_idx : source
metabib_keyword_field_entry_value_idx : "substring"(value, 1, 1024)) WHERE ((index_vector)::pg_catalog.tsvector = (''::pg_catalog.tsvector)::tsvector)::pg_catalog.tsvector

Table: metarecord

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
fingerprint : text -- NOT NULL,
master_record : bigint -- REFERENCES biblio.record_entry.
mods : text --

Indexes:

metabib_metarecord_fingerprint_idx : fingerprint
metabib_metarecord_master_record_idx : master_record

Tables referencing metabib.metarecord_source_map via Foreign Key Constraints:

metabib.metarecord_source_map

Table: metarecord_source_map

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
metarecord : bigint -- NOT NULL, REFERENCES metabib.metarecord.
source : bigint -- NOT NULL, REFERENCES biblio.record_entry.

Indexes:

metabib_metarecord_source_map_metarecord_idx : metarecord
metabib_metarecord_source_map_source_record_idx : source

Table: real_full_rec

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('metabib.full_rec_id_seq'::regclass),

record : bigint -- NOT NULL, REFERENCES biblio.record_entry.
tag : character(3) -- NOT NULL,
ind1 : text --
ind2 : text --
subfield : text --
value : text -- NOT NULL,
index_vector : tsvector -- NOT NULL,

Indexes:

metabib_full_rec_index_vector_idx : index_vector
metabib_full_rec_record_idx : record
metabib_full_rec_tag_subfield_idx : tag, subfield
metabib_full_rec_value_idx : "substring"(value, 1, 1024)
metabib_full_rec_value_tpo_index : "substring"(value, 1, 1024) text_pattern_ops

Table: rec_descriptor

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
record : bigint -- REFERENCES biblio.record_entry.
item_type : text --
item_form : text --
bib_level : text --
control_type : text --
char_encoding : text --
enc_level : text --
audience : text --
lit_form : text --
type_mat : text --
cat_form : text --
pub_status : text --
item_lang : text --
vr_format : text --
date1 : text --
date2 : text --

Indexes:

metabib_rec_descriptor_audience_idx : audience
metabib_rec_descriptor_bib_level_idx : bib_level
metabib_rec_descriptor_cat_form_idx : cat_form
metabib_rec_descriptor_char_encoding_idx : char_encoding
metabib_rec_descriptor_control_type_idx : control_type
metabib_rec_descriptor_date1_idx : date1
metabib_rec_descriptor_dates_idx : date1, date2
metabib_rec_descriptor_enc_level_idx : enc_level
metabib_rec_descriptor_item_form_idx : item_form
metabib_rec_descriptor_item_lang_idx : item_lang

metabib_rec_descriptor_item_type_idx : item_type
metabib_rec_descriptor_lit_form_idx : lit_form
metabib_rec_descriptor_pub_status_idx : pub_status
metabib_rec_descriptor_record_idx : record
metabib_rec_descriptor_vr_format_idx : vr_format

Table: series_field_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
source : bigint -- `NOT NULL`, `REFERENCES` biblio.record_entry.
field : integer -- `NOT NULL`, `REFERENCES` config.metabib_field.
value : text -- `NOT NULL`,
index_vector : tsvector -- `NOT NULL`,

Indexes:

metabib_series_field_entry_index_vector_idx : index_vector
metabib_series_field_entry_source_idx : source
metabib_series_field_entry_value_idx : "substring"(value, 1, 1024)) WHERE ((index_vector)::pg_catalog.tsvector = (''::pg_catalog.tsvector)::tsvector)::pg_catalog.tsvector

Table: subject_field_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
source : bigint -- `NOT NULL`, `REFERENCES` biblio.record_entry.
field : integer -- `NOT NULL`, `REFERENCES` config.metabib_field.
value : text -- `NOT NULL`,
index_vector : tsvector -- `NOT NULL`,

Indexes:

metabib_subject_field_entry_index_vector_idx : index_vector
metabib_subject_field_entry_source_idx : source
metabib_subject_field_entry_value_idx : "substring"(value, 1, 1024)) WHERE ((index_vector)::pg_catalog.tsvector = (''::pg_catalog.tsvector)::tsvector)::pg_catalog.tsvector

Table: title_field_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
source : bigint -- `NOT NULL`, `REFERENCES` biblio.record_entry.
field : integer -- `NOT NULL`, `REFERENCES` config.metabib_field.
value : text -- `NOT NULL`,

index_vector : tsvector -- NOT NULL,

Indexes:

metabib_title_field_entry_index_vector_idx : index_vector
metabib_title_field_entry_source_idx : source
metabib_title_field_entry_value_idx : "substring"(value, 1, 1024)) WHERE ((index_vector)::pg_catalog.tsvector = (("::pg_catalog.tsvector)::tsvector)::pg_catalog.tsvector

reingest_metabib_field_entries(bib_id bigint)

Function Properties

Language: PLPGSQL
Return Type: void

reingest_metabib_full_rec(bib_id bigint)

Function Properties

Language: PLPGSQL
Return Type: void

reingest_metabib_rec_descriptor(bib_id bigint)

Function Properties

Language: PLPGSQL
Return Type: void

remap_metarecord_for_bib(fp bigint, bib_id text)

Function Properties

Language: PLPGSQL
Return Type: bigint

Schema money

Table: billable_xact

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
usr : integer -- NOT NULL, REFERENCES actor.usr.
xact_start : timestamp with time zone -- NOT NULL, DEFAULT now(),
xact_finish : timestamp with time zone --

unrecovered : boolean --

Indexes:

m_b_x_open_xacts_idx : usr

View: billable_xact_summary

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
usr : integer --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
total_paid : numeric --
last_payment_ts : timestamp with time zone --
last_payment_note : text --
last_payment_type : name --
total_owed : numeric --
last_billing_ts : timestamp with time zone --
last_billing_note : text --
last_billing_type : text --
balance_owed : numeric --
xact_type : name --

View: billable_xact_summary_location_view

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
usr : integer --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
total_paid : numeric --
last_payment_ts : timestamp with time zone --
last_payment_note : text --
last_payment_type : name --
total_owed : numeric --
last_billing_ts : timestamp with time zone --
last_billing_note : text --
last_billing_type : text --
balance_owed : numeric --
xact_type : name --
billing_location : integer --

View: billable_xact_with_void_summary

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
usr : integer --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
total_paid : numeric --
last_payment_ts : timestamp with time zone --
last_payment_note : text --
last_payment_type : name --
total_owed : numeric --
last_billing_ts : timestamp with time zone --
last_billing_note : text --
last_billing_type : text --
balance_owed : numeric --
xact_type : name --

Table: billing

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
xact : bigint -- `NOT NULL`,
billing_ts : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
voided : boolean -- `NOT NULL`, `DEFAULT false`,
voider : integer --
void_time : timestamp with time zone --
amount : numeric(6,2) -- `NOT NULL`,
billing_type : text -- `NOT NULL`,
btype : integer -- `NOT NULL`, `REFERENCES` config.billing_type.
note : text --

Indexes:

m_b_time_idx : billing_ts
m_b_xact_idx : xact

Table: bnm_desk_payment

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`, `DEFAULT nextval('money.payment_id_seq'::regclass)`,
xact : bigint -- `NOT NULL`,
payment_ts : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
voided : boolean -- `NOT NULL`, `DEFAULT false`,
amount : numeric(6,2) -- `NOT NULL`,
note : text --
amount_collected : numeric(6,2) -- `NOT NULL`,
accepting_usr : integer -- `NOT NULL`,
cash_drawer : integer -- `REFERENCES` actor.workstation.

Table: bnm_payment

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('money.payment_id_seq'::regclass),
xact : bigint -- NOT NULL,
payment_ts : timestamp with time zone -- NOT NULL, DEFAULT now(),
voided : boolean -- NOT NULL, DEFAULT false,
amount : numeric(6,2) -- NOT NULL,
note : text --
amount_collected : numeric(6,2) -- NOT NULL,
accepting_usr : integer -- NOT NULL,

View: bnm_payment_view

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
xact : bigint --
payment_ts : timestamp with time zone --
voided : boolean --
amount : numeric(6,2) --
note : text --
amount_collected : numeric(6,2) --
accepting_usr : integer --
payment_type : name --

Table: cash_payment

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('money.payment_id_seq'::regclass),
xact : bigint -- NOT NULL,
payment_ts : timestamp with time zone -- NOT NULL, DEFAULT now(),
voided : boolean -- NOT NULL, DEFAULT false,
amount : numeric(6,2) -- NOT NULL,
note : text --
amount_collected : numeric(6,2) -- NOT NULL,
accepting_usr : integer -- NOT NULL,
cash_drawer : integer --

Indexes:

money_cash_id_idx : id
money_cash_payment_accepting_usr_idx : accepting_usr
money_cash_payment_cash_drawer_idx : cash_drawer
money_cash_payment_ts_idx : payment_ts

money_cash_payment_xact_idx : xact

View: cashdrawer_payment_view

Columns:

field name : datatype -- parameters, contraints and notes
org_unit : integer --
cashdrawer : integer --
payment_type : name --
payment_ts : timestamp with time zone --
amount : numeric(6,2) --
voided : boolean --
note : text --

Table: check_payment

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`, `DEFAULT nextval('money.payment_id_seq'::regclass)`,
xact : bigint -- `NOT NULL`,
payment_ts : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
voided : boolean -- `NOT NULL`, `DEFAULT false`,
amount : numeric(6,2) -- `NOT NULL`,
note : text --
amount_collected : numeric(6,2) -- `NOT NULL`,
accepting_usr : integer -- `NOT NULL`,
cash_drawer : integer --
check_number : text -- `NOT NULL`,

Indexes:

money_check_id_idx : id
money_check_payment_accepting_usr_idx : accepting_usr
money_check_payment_cash_drawer_idx : cash_drawer
money_check_payment_ts_idx : payment_ts
money_check_payment_xact_idx : xact

Table: collections_tracker

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
usr : integer -- `NOT NULL`, `REFERENCES` actor.usr.
collector : integer -- `NOT NULL`, `REFERENCES` actor.usr.
location : integer -- `NOT NULL`, `REFERENCES` actor.org_unit.
enter_time : timestamp with time zone --

Indexes:

m_c_t_collector_idx : collector

Table: credit_card_payment

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`, `DEFAULT nextval('money.payment_id_seq'::regclass)`,
xact : bigint -- `NOT NULL`,
payment_ts : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
voided : boolean -- `NOT NULL`, `DEFAULT false`,
amount : numeric(6,2) -- `NOT NULL`,
note : text --
amount_collected : numeric(6,2) -- `NOT NULL`,
accepting_usr : integer -- `NOT NULL`,
cash_drawer : integer --
cc_type : text --
cc_number : text --
cc_processor : text --
cc_first_name : text --
cc_last_name : text --
expire_month : integer --
expire_year : integer --
approval_code : text --

Indexes:

money_credit_card_id_idx : id
money_credit_card_payment_accepting_usr_idx : accepting_usr
money_credit_card_payment_cash_drawer_idx : cash_drawer
money_credit_card_payment_ts_idx : payment_ts
money_credit_card_payment_xact_idx : xact

Table: credit_payment

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`, `DEFAULT nextval('money.payment_id_seq'::regclass)`,
xact : bigint -- `NOT NULL`,
payment_ts : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
voided : boolean -- `NOT NULL`, `DEFAULT false`,
amount : numeric(6,2) -- `NOT NULL`,
note : text --
amount_collected : numeric(6,2) -- `NOT NULL`,
accepting_usr : integer -- `NOT NULL`,

Indexes:

money_credit_id_idx : id
money_credit_payment_accepting_usr_idx : accepting_usr

money_credit_payment_payment_ts_idx : payment_ts
money_credit_payment_xact_idx : xact

View: desk_payment_view

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
xact : bigint --
payment_ts : timestamp with time zone --
voided : boolean --
amount : numeric(6,2) --
note : text --
amount_collected : numeric(6,2) --
accepting_usr : integer --
cash_drawer : integer --
payment_type : name --

Table: forgive_payment

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`, `DEFAULT nextval('money.payment_id_seq'::regclass)`,
xact : bigint -- `NOT NULL`,
payment_ts : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
voided : boolean -- `NOT NULL`, `DEFAULT false`,
amount : numeric(6,2) -- `NOT NULL`,
note : text --
amount_collected : numeric(6,2) -- `NOT NULL`,
accepting_usr : integer -- `NOT NULL`,

Indexes:

money_forgive_id_idx : id
money_forgive_payment_accepting_usr_idx : accepting_usr
money_forgive_payment_payment_ts_idx : payment_ts
money_forgive_payment_xact_idx : xact

Table: goods_payment

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`, `DEFAULT nextval('money.payment_id_seq'::regclass)`,
xact : bigint -- `NOT NULL`,
payment_ts : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
voided : boolean -- `NOT NULL`, `DEFAULT false`,
amount : numeric(6,2) -- `NOT NULL`,
note : text --

amount_collected : numeric(6,2) -- NOT NULL,
accepting_usr : integer -- NOT NULL,

Indexes:

money_goods_id_idx : id
money_goods_payment_accepting_usr_idx : accepting_usr
money_goods_payment_payment_ts_idx : payment_ts
money_goods_payment_xact_idx : xact

Table: grocery

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('money.billable_xact_id_seq'::regclass),
usr : integer -- NOT NULL,
xact_start : timestamp with time zone -- NOT NULL, DEFAULT now(),
xact_finish : timestamp with time zone --
unrecovered : boolean --
billing_location : integer -- NOT NULL,
note : text --

Indexes:

circ_open_date_idx : xact_start) WHERE (xact_finish IS NULL
m_g_usr_idx : usr

Table: materialized_billable_xact_summary

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY,
usr : integer --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
total_paid : numeric --
last_payment_ts : timestamp with time zone --
last_payment_note : text --
last_payment_type : name --
total_owed : numeric --
last_billing_ts : timestamp with time zone --
last_billing_note : text --
last_billing_type : text --
balance_owed : numeric --
xact_type : name --

Indexes:

money_mat_summary_usr_idx : usr
money_mat_summary_xact_start_idx : xact_start

View: non_drawer_payment_view

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
xact : bigint --
payment_ts : timestamp with time zone --
voided : boolean --
amount : numeric(6,2) --
note : text --
amount_collected : numeric(6,2) --
accepting_usr : integer --
payment_type : name --

View: open_billable_xact_summary

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
usr : integer --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
total_paid : numeric --
last_payment_ts : timestamp with time zone --
last_payment_note : text --
last_payment_type : name --
total_owed : numeric --
last_billing_ts : timestamp with time zone --
last_billing_note : text --
last_billing_type : text --
balance_owed : numeric --
xact_type : name --
billing_location : integer --

View: open_transaction_billing_summary

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
last_billing_type : text --
last_billing_note : text --
last_billing_ts : timestamp with time zone --
total_owed : numeric --

View: open_transaction_billing_type_summary

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
last_billing_type : text --
last_billing_note : text --
last_billing_ts : timestamp with time zone --
total_owed : numeric --

View: open_transaction_payment_summary

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
last_payment_type : name --
last_payment_note : text --
last_payment_ts : timestamp with time zone --
total_paid : numeric --

View: open_usr_circulation_summary

Columns:

field name : datatype -- parameters, contraints and notes
usr : integer --
total_paid : numeric --
total_owed : numeric --
balance_owed : numeric --

View: open_usr_summary

Columns:

field name : datatype -- parameters, contraints and notes
usr : integer --
total_paid : numeric --
total_owed : numeric --
balance_owed : numeric --

Table: payment

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
xact : bigint -- `NOT NULL`,
payment_ts : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
voided : boolean -- `NOT NULL`, `DEFAULT false`,
amount : numeric(6,2) -- `NOT NULL`,
note : text --

Indexes:

m_p_time_idx : payment_ts
m_p_xact_idx : xact

View: payment_view

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
xact : bigint --
payment_ts : timestamp with time zone --
voided : boolean --
amount : numeric(6,2) --
note : text --
payment_type : name --

View: transaction_billing_summary

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
last_billing_type : text --
last_billing_note : text --
last_billing_ts : timestamp with time zone --
total_owed : numeric --

View: transaction_billing_type_summary

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
last_billing_type : text --
last_billing_note : text --
last_billing_ts : timestamp with time zone --
total_owed : numeric --

View: transaction_billing_with_void_summary

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
last_billing_type : text --
last_billing_note : text --
last_billing_ts : timestamp with time zone --

total_owed : numeric --

View: transaction_payment_summary

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
last_payment_type : name --
last_payment_note : text --
last_payment_ts : timestamp with time zone --
total_paid : numeric --

View: transaction_payment_with_void_summary

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
last_payment_type : name --
last_payment_note : text --
last_payment_ts : timestamp with time zone --
total_paid : numeric --

View: usr_circulation_summary

Columns:

field name : datatype -- parameters, contraints and notes
usr : integer --
total_paid : numeric --
total_owed : numeric --
balance_owed : numeric --

View: usr_summary

Columns:

field name : datatype -- parameters, contraints and notes
usr : integer --
total_paid : numeric --
total_owed : numeric --
balance_owed : numeric --

Table: work_payment

Columns:

field name : datatype -- parameters, contraints and notes

id : bigint -- PRIMARY KEY, DEFAULT nextval('money.payment_id_seq'::regclass),
xact : bigint -- NOT NULL,
payment_ts : timestamp with time zone -- NOT NULL, DEFAULT now(),
voided : boolean -- NOT NULL, DEFAULT false,
amount : numeric(6,2) -- NOT NULL,
note : text --
amount_collected : numeric(6,2) -- NOT NULL,
accepting_usr : integer -- NOT NULL,

Indexes:

money_work_id_idx : id
money_work_payment_accepting_usr_idx : accepting_usr
money_work_payment_payment_ts_idx : payment_ts
money_work_payment_xact_idx : xact

mat_summary_create()

Function Properties

Language: PLPGSQL
Return Type: trigger

mat_summary_delete()

Function Properties

Language: PLPGSQL
Return Type: trigger

mat_summary_update()

Function Properties

Language: PLPGSQL
Return Type: trigger

materialized_summary_billing_add()

Function Properties

Language: PLPGSQL
Return Type: trigger

materialized_summary_billing_del()

Function Properties

Language: PLPGSQL

Return Type: trigger

materialized_summary_billing_update()

Function Properties

Language: PLPGSQL
Return Type: trigger

materialized_summary_payment_add()

Function Properties

Language: PLPGSQL
Return Type: trigger

materialized_summary_payment_del()

Function Properties

Language: PLPGSQL
Return Type: trigger

materialized_summary_payment_update()

Function Properties

Language: PLPGSQL
Return Type: trigger

Schema offline

Table: script

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
session : text -- `NOT NULL`,
requestor : integer -- `NOT NULL`,
create_time : integer -- `NOT NULL`,
workstation : text -- `NOT NULL`,
logfile : text -- `NOT NULL`,
time_delta : integer -- `NOT NULL`,
count : integer -- `NOT NULL`,

Indexes:

offline_script_pkey : id
offline_script_session : session
offline_script_ws : workstation

Table: session

Columns:

field name : datatype -- parameters, contraints and notes
key : text -- PRIMARY KEY,
org : integer -- NOT NULL,
description : text --
creator : integer -- NOT NULL,
create_time : integer -- NOT NULL,
in_process : integer -- NOT NULL,
start_time : integer --
end_time : integer --
num_complete : integer -- NOT NULL,

Indexes:

offline_session_creation : create_time
offline_session_org : org
offline_session_pkey : key

Schema permission

Table: grp_penalty_threshold

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
grp : integer -- UNIQUE#1, NOT NULL, REFERENCES permission.grp_tree.
org_unit : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
penalty : integer -- UNIQUE#1, NOT NULL, REFERENCES config.standing_penalty.
threshold : numeric(8,2) -- NOT NULL,

Table: grp_perm_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
grp : integer -- UNIQUE#1, NOT NULL, REFERENCES permission.grp_tree.
perm : integer -- UNIQUE#1, NOT NULL, REFERENCES permission.perm_list.
depth : integer -- NOT NULL,
grantable : boolean -- NOT NULL, DEFAULT false,

Table: grp_tree

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
name : text -- UNIQUE, NOT NULL,
parent : integer -- REFERENCES permission.grp_tree.
usergroup : boolean -- NOT NULL, DEFAULT true,
perm_interval : interval -- NOT NULL, DEFAULT '3 years'::interval,
description : text --
application_perm : text --

Indexes:

grp_tree_parent_idx : parent

Tables referencing actor.usr via Foreign Key Constraints:

actor.usr
config.circ_matrix_matchpoint
config.hold_matrix_matchpoint
permission.grp_penalty_threshold
permission.grp_perm_map
permission.grp_tree
permission.usr_grp_map

Table: perm_list

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
code : text -- UNIQUE, NOT NULL,
description : text --

Indexes:

perm_list_code_idx : code

Tables referencing config.org_unit_setting_type via Foreign Key Constraints:

config.org_unit_setting_type
permission.grp_perm_map
permission.usr_object_perm_map
permission.usr_perm_map

Table: usr_grp_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
usr : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.usr.
grp : integer -- UNIQUE#1, NOT NULL, REFERENCES permission.grp_tree.

Table: usr_object_perm_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
usr : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.usr.
perm : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` permission.perm_list.
object_type : text -- `UNIQUE#1`, `NOT NULL`,
object_id : text -- `UNIQUE#1`, `NOT NULL`,
grantable : boolean -- `NOT NULL`, `DEFAULT false`,

Indexes:

uopm_usr_idx : usr

Table: usr_perm_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
usr : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.usr.
perm : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` permission.perm_list.
depth : integer -- `NOT NULL`,
grantable : boolean -- `NOT NULL`, `DEFAULT false`,

Table: usr_work_ou_map

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
usr : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.usr.
work_ou : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.org_unit.

grp_ancestors(integer)

Function Properties

Language: SQL
Return Type: SET OF grp_tree

grp_ancestors_distance(distance integer)

Function Properties

Language: SQL

Return Type: SET OF record

grp_descendants_distance(distance integer)

Function Properties

Language: SQL
Return Type: SET OF record

usr_can_grant_perm(target_ou integer, tperm text, iuser integer)

Function Properties

Language: PLPGSQL
Return Type: boolean

usr_has_home_perm(target_ou integer, tperm text, iuser integer)

Function Properties

Language: PLPGSQL
Return Type: boolean

usr_has_object_perm(integer, text, text, text)

Function Properties

Language: SQL
Return Type: boolean

usr_has_object_perm(target_ou integer, obj_id text, obj_type text, tperm text, iuser integer)

Function Properties

Language: PLPGSQL
Return Type: boolean

usr_has_perm(integer, text, integer)

Function Properties

Language: SQL
Return Type: boolean

usr_has_perm_at(perm_code integer, user_id text)

Function Properties

Language: SQL
Return Type: SET OF integer

usr_has_perm_at_all(perm_code integer, user_id text)

Function Properties

Language: SQL
Return Type: SET OF integer

usr_has_perm_at_all_nd(perm_code integer, user_id text)

Function Properties

Language: PLPGSQL
Return Type: SET OF integer

usr_has_perm_at_nd(perm_code integer, user_id text)

Function Properties

Language: PLPGSQL
Return Type: SET OF integer

usr_has_work_perm(target_ou integer, tperm text, iuser integer)

Function Properties

Language: PLPGSQL
Return Type: boolean

usr_perms(integer)

Function Properties

Language: SQL
Return Type: SET OF usr_perm_map

Schema public

standard public schema

Table: staging_items

Columns:

field name : datatype -- parameters, contraints and notes
l_callnum : text --
hseq : integer --
egid : integer --
createdate : date --
l_location : text --
l_barcode : text --
l_circ_modifier : text --
l_owning_lib : text --

_get_parser_from_curcfg()

Function Properties

Language: SQL
Return Type: text

agg_text(text)

Function Properties

Language: INTERNAL
Return Type: text

agg_tsvector(tsvector)

Function Properties

Language: INTERNAL
Return Type: tsvector

call_number_dewey(text)

Function Properties

Language: PLPERLU
Return Type: text

call_number_dewey(text, integer)

Function Properties

Language: SQL
Return Type: text

cleanup_acq_marc()

Function Properties

Language: PLPGSQL
Return Type: trigger

concat(tsvector, tsvector)

Function Properties

Language: INTERNAL
Return Type: tsvector

connectby(text, text, text, text, integer)

Function Properties

Language: C
Return Type: SET OF record

connectby(text, text, text, text, integer, text)

Function Properties

Language: C
Return Type: SET OF record

connectby(text, text, text, text, text, integer)

Function Properties

Language: C
Return Type: SET OF record

connectby(text, text, text, text, text, integer, text)

Function Properties

Language: C
Return Type: SET OF record

crosstab(text)

Function Properties

Language: C
Return Type: SET OF record

crosstab(text, integer)

Function Properties

Language: C
Return Type: SET OF record

crosstab(text, text)

Function Properties

Language: C
Return Type: SET OF record

crosstab2(text)

Function Properties

Language: C
Return Type: SET OF tablefunc_crosstab_2

crosstab3(text)

Function Properties

Language: C
Return Type: SET OF tablefunc_crosstab_3

crosstab4(text)

Function Properties

Language: C
Return Type: SET OF tablefunc_crosstab_4

dex_init(internal)

Function Properties

Language: C
Return Type: internal

dex_lexize(internal, internal, integer)

Function Properties

Language: C
Return Type: internal

entityize(text)

Function Properties

Language: PLPERLU
Return Type: text

explode_array(anyarray)

Function Properties

Language: SQL
Return Type: SET OF anyelement

extract_acq_marc_field(bigint, text, text)

Function Properties

Language: SQL
Return Type: text

extract_marc_field(text, bigint, text)

Function Properties

Language: SQL
Return Type: text

extract_marc_field(text, bigint, text, text)

Function Properties

Language: PLPGSQL
Return Type: text

facet_force_nfc()

Function Properties

Language: PLPGSQL
Return Type: trigger

first(anyelement)

Function Properties

Language: INTERNAL
Return Type: anyelement

first5(text)

Function Properties

Language: SQL
Return Type: text

first_agg(anyelement, anyelement)

Function Properties

Language: SQL
Return Type: anyelement

first_word(text)

Function Properties

Language: SQL
Return Type: text

force_unicode_normal_form(form text, string text)

Function Properties

Language: PLPERLU
Return Type: text

get_covers(tsvector, tsquery)

Function Properties

Language: C
Return Type: text

headline(oid, text, tsquery)

Function Properties

Language: INTERNAL
Return Type: text

headline(oid, text, tsquery, text)

Function Properties

Language: INTERNAL
Return Type: text

headline(text, text, tsquery)

Function Properties

Language: C
Return Type: text

headline(text, text, tsquery, text)

Function Properties

Language: C
Return Type: text

headline(text, tsquery)

Function Properties

Language: INTERNAL
Return Type: text

headline(text, tsquery, text)

Function Properties

Language: INTERNAL
Return Type: text

ingest_acq_marc()

Function Properties

Language: PLPGSQL
Return Type: trigger

is_json(text)

Function Properties

Language: PLPERLU
Return Type: boolean

last(anyelement)

Function Properties

Language: INTERNAL
Return Type: anyelement

last_agg(anyelement, anyelement)

Function Properties

Language: SQL
Return Type: anyelement

left_trunc(text, integer)

Function Properties

Language: SQL
Return Type: text

length(tsvector)

Function Properties

Language: INTERNAL
Return Type: integer

lexize(oid, text)

Function Properties

Language: INTERNAL
Return Type: text[]

lexize(text)

Function Properties

Language: C
Return Type: text[]

lexize(text, text)

Function Properties

Language: C
Return Type: text[]

lowercase(text)

Function Properties

Language: PLPERLU
Return Type: text

maintain_901()

Function Properties

Language: PLPGSQL
Return Type: trigger

maintain_control_numbers()

Function Properties

Language: PLPERLU
Return Type: trigger

naco_normalize(text)

Function Properties

Language: SQL
Return Type: text

naco_normalize(text, text)

Function Properties

Language: PLPERLU
Return Type: text

naco_normalize_keep_comma(text)

Function Properties

Language: SQL
Return Type: text

non_filing_normalize(text, "char")

Function Properties

Language: SQL
Return Type: text

normal_rand(integer, double precision, double precision)

Function Properties

Language: C
Return Type: SET OF double precision

normalize_space(text)

Function Properties

Language: SQL
Return Type: text

numnode(tsquery)

Function Properties

Language: INTERNAL
Return Type: integer

oils_i18n_code_tracking()

Function Properties

Language: PLPGSQL
Return Type: trigger

oils_i18n_gettext(integer, text, text, text)

Function Properties

Language: SQL
Return Type: text

oils_i18n_gettext(text, text, text, text)

Function Properties

Language: SQL
Return Type: text

oils_i18n_id_tracking()

Function Properties

Language: PLPGSQL
Return Type: trigger

oils_i18n_update_apply(hint text, new_ident text, old_ident text)

Function Properties

Language: PLPGSQL
Return Type: void

oils_i18n_xlate(raw_locale text, keyvalue text, identcol text, keycol text, keyclass text, keytable text)

Function Properties

Language: PLPGSQL
Return Type: text

oils_json_to_text(text)

Function Properties

Language: PLPERLU
Return Type: text

oils_text_as_bytea(text)

Function Properties

Language: SQL
Return Type: bytea

oils_tsearch2()

Function Properties

Language: PLPGSQL
Return Type: trigger

oils_xpath(text, text)

Function Properties

Language: SQL
Return Type: text[]

oils_xpath(text, text, anyarray)

Function Properties

Language: SQL
Return Type: text[]

oils_xpath_string(text, text)

Function Properties

Language: SQL
Return Type: text

oils_xpath_string(text, text, anyarray)

Function Properties

Language: SQL
Return Type: text

oils_xpath_string(text, text, text)

Function Properties

Language: SQL
Return Type: text

oils_xpath_string(text, text, text, anyarray)

Function Properties

Language: SQL
Return Type: text

oils_xpath_table(criteria text, xpaths text, relation_name text, document_field text, key text)

Function Properties

Language: PLPGSQL
Return Type: SET OF record

oils_xslt_process(text, text)

Function Properties

Language: PLPERLU
Return Type: text

parse(oid, text)

Function Properties

Language: INTERNAL
Return Type: SET OF tokenout

parse(text)

Function Properties

Language: C

Return Type: SET OF tokenout

parse(text, text)

Function Properties

Language: INTERNAL
Return Type: SET OF tokenout

plainto_tsquery(oid, text)

Function Properties

Language: INTERNAL
Return Type: tsquery

plainto_tsquery(text)

Function Properties

Language: INTERNAL
Return Type: tsquery

plainto_tsquery(text, text)

Function Properties

Language: C
Return Type: tsquery

prsd_end(internal)

Function Properties

Language: C
Return Type: void

prsd_getlexeme(internal, internal, internal)

Function Properties

Language: C
Return Type: integer

prsd_headline(internal, internal, internal)

Function Properties

Language: C

Return Type: internal

prsd_lextype(internal)

Function Properties

Language: C
Return Type: internal

prsd_start(internal, integer)

Function Properties

Language: C
Return Type: internal

querytree(tsquery)

Function Properties

Language: INTERNAL
Return Type: text

rank(real[], tsvector, tsquery)

Function Properties

Language: INTERNAL
Return Type: real

rank(real[], tsvector, tsquery, integer)

Function Properties

Language: INTERNAL
Return Type: real

rank(tsvector, tsquery)

Function Properties

Language: INTERNAL
Return Type: real

rank(tsvector, tsquery, integer)

Function Properties

Language: INTERNAL

Return Type: real

rank_cd(real[], tsvector, tsquery)

Function Properties

Language: INTERNAL
Return Type: real

rank_cd(real[], tsvector, tsquery, integer)

Function Properties

Language: INTERNAL
Return Type: real

rank_cd(tsvector, tsquery)

Function Properties

Language: INTERNAL
Return Type: real

rank_cd(tsvector, tsquery, integer)

Function Properties

Language: INTERNAL
Return Type: real

remove_commas(text)

Function Properties

Language: SQL
Return Type: text

remove_diacritics(text)

Function Properties

Language: PLPERLU
Return Type: text

remove_paren_substring(text)

Function Properties

Language: SQL

Return Type: text

remove_whitespace(text)

Function Properties

Language: SQL
Return Type: text

reset_tsearch()

Function Properties

Language: C
Return Type: void

rewrite(tsquery, text)

Function Properties

Language: INTERNAL
Return Type: tsquery

rewrite(tsquery, tsquery, tsquery)

Function Properties

Language: INTERNAL
Return Type: tsquery

rewrite(tsquery[])

Function Properties

Language: INTERNAL
Return Type: tsquery

rewrite_accum(tsquery, tsquery[])

Function Properties

Language: C
Return Type: tsquery

rewrite_finish(tsquery)

Function Properties

Language: C

Return Type: tsquery

right_trunc(text, integer)

Function Properties

Language: SQL
Return Type: text

set_curcfg(integer)

Function Properties

Language: C
Return Type: void

set_curcfg(text)

Function Properties

Language: C
Return Type: void

set_curdict(integer)

Function Properties

Language: C
Return Type: void

set_curdict(text)

Function Properties

Language: C
Return Type: void

set_curprs(integer)

Function Properties

Language: C
Return Type: void

set_curprs(text)

Function Properties

Language: C

Return Type: void

setweight(tsvector, "char")

Function Properties

Language: INTERNAL
Return Type: tsvector

show_curcfg()

Function Properties

Language: INTERNAL
Return Type: oid

snb_en_init(internal)

Function Properties

Language: C
Return Type: internal

snb_lexize(internal, internal, integer)

Function Properties

Language: C
Return Type: internal

snb_ru_init(internal)

Function Properties

Language: C
Return Type: internal

snb_ru_init_koi8(internal)

Function Properties

Language: C
Return Type: internal

snb_ru_init_utf8(internal)

Function Properties

Language: C

Return Type: internal

spell_init(internal)

Function Properties

Language: C
Return Type: internal

spell_lexize(internal, internal, integer)

Function Properties

Language: C
Return Type: internal

split_date_range(text)

Function Properties

Language: SQL
Return Type: text

stat(text)

Function Properties

Language: INTERNAL
Return Type: SET OF statinfo

stat(text, text)

Function Properties

Language: INTERNAL
Return Type: SET OF statinfo

strip(tsvector)

Function Properties

Language: INTERNAL
Return Type: tsvector

syn_init(internal)

Function Properties

Language: C

Return Type: internal

syn_lexize(internal, internal, integer)

Function Properties

Language: C
Return Type: internal

tableoid2name(oid)

Function Properties

Language: PLPGSQL
Return Type: text

text_concat(text, text)

Function Properties

Language: SQL
Return Type: text

thesaurus_init(internal)

Function Properties

Language: C
Return Type: internal

thesaurus_lexize(internal, internal, integer, internal)

Function Properties

Language: C
Return Type: internal

to_tsquery(oid, text)

Function Properties

Language: INTERNAL
Return Type: tsquery

to_tsquery(text)

Function Properties

Language: INTERNAL

Return Type: tsquery

to_tsquery(text, text)

Function Properties

Language: C
Return Type: tsquery

to_tsvector(oid, text)

Function Properties

Language: INTERNAL
Return Type: tsvector

to_tsvector(text)

Function Properties

Language: INTERNAL
Return Type: tsvector

to_tsvector(text, text)

Function Properties

Language: C
Return Type: tsvector

token_type()

Function Properties

Language: C
Return Type: SET OF tokentype

token_type(integer)

Function Properties

Language: INTERNAL
Return Type: SET OF tokentype

token_type(text)

Function Properties

Language: INTERNAL

Return Type: SET OF tokentype

translate_isbn1013(text)

The translate_isbn1013 function takes an input ISBN and returns the following in a single space-delimited string if the input ISBN is valid: - The normalized input ISBN (hyphens stripped) - The normalized input ISBN with a fixed checksum if the checksum was bad - The ISBN converted to its ISBN10 or ISBN13 counterpart, if possible

Function Properties

Language: PLPERLU
Return Type: text

ts_debug(text)

Function Properties

Language: SQL
Return Type: SET OF tsdebug

tsearch2()

Function Properties

Language: C
Return Type: trigger

tsq_mcontained(tsquery, tsquery)

Function Properties

Language: INTERNAL
Return Type: boolean

tsq_mcontains(tsquery, tsquery)

Function Properties

Language: INTERNAL
Return Type: boolean

tsquery_and(tsquery, tsquery)

Function Properties

Language: INTERNAL
Return Type: tsquery

tsquery_not(tsquery)

Function Properties

Language: INTERNAL
Return Type: tsquery

tsquery_or(tsquery, tsquery)

Function Properties

Language: INTERNAL
Return Type: tsquery

tsvector_concat(tsvector, tsvector)

Function Properties

Language: SQL
Return Type: tsvector

uppercase(text)

Function Properties

Language: PLPERLU
Return Type: text

xml_encode_special_chars(text)

Function Properties

Language: C
Return Type: text

xml_is_well_formed(text)

Function Properties

Language: C
Return Type: boolean

xml_valid(text)

Function Properties

Language: C
Return Type: boolean

xpath_bool(text, text)

Function Properties

Language: C
Return Type: boolean

xpath_list(text, text)

Function Properties

Language: SQL
Return Type: text

xpath_list(text, text, text)

Function Properties

Language: C
Return Type: text

xpath_nodeset(text, text)

Function Properties

Language: SQL
Return Type: text

xpath_nodeset(text, text, text)

Function Properties

Language: SQL
Return Type: text

xpath_nodeset(text, text, text, text)

Function Properties

Language: C
Return Type: text

xpath_number(text, text)

Function Properties

Language: C

Return Type: real

xpath_string(text, text)

Function Properties

Language: C
Return Type: text

xpath_table(text, text, text, text, text)

Function Properties

Language: C
Return Type: SET OF record

xslt_process(text, text)

Function Properties

Language: C
Return Type: text

xslt_process(text, text, text)

Function Properties

Language: C
Return Type: text

Schema query

Contains tables designed to represent user-defined queries for reports and the like.

Table: bind_variable

Columns:

field name : datatype -- parameters, contraints and notes
name : text -- `PRIMARY KEY`,
type : text -- `NOT NULL`,
description : text -- `NOT NULL`,
default_value : text --
label : text -- `NOT NULL`,

Constraints:

bind_variable_type : CHECK ((type = ANY (ARRAY['string'::text, 'number'::text, 'string_list'::text, 'number_list'::text])))

Tables referencing query.expression via Foreign Key Constraints:

query.expression

Table: case_branch

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
parent_expr : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` query.expression.
seq_no : integer -- `UNIQUE#1`, `NOT NULL`,
condition : integer -- `REFERENCES` query.expression.
result : integer -- `NOT NULL`, `REFERENCES` query.expression.

Table: datatype

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
datatype_name : text -- `UNIQUE`, `NOT NULL`,
is_numeric : boolean -- `NOT NULL`, `DEFAULT false`,
is_composite : boolean -- `NOT NULL`, `DEFAULT false`,

Constraints:

qdt_comp_not_num : CHECK (((is_numeric IS FALSE) OR (is_composite IS FALSE)))

Tables referencing query.expression via Foreign Key Constraints:

query.expression
query.function_param_def
query.function_sig
query.record_column
query.subfield

View: expr_xbet

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
left_operand : integer --
negate : boolean --

View: expr_xbind

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
bind_variable : text --

View: expr_xbool

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
literal : text --
negate : boolean --

View: expr_xcase

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
left_operand : integer --
negate : boolean --

View: expr_xcast

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
left_operand : integer --
cast_type : integer --
negate : boolean --

View: expr_xcol

Columns:

field name : datatype -- parameters, contraints and notes

id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
table_alias : text --
column_name : text --
negate : boolean --

View: expr_xex

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
subquery : integer --
negate : boolean --

View: expr_xfunc

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
column_name : text --
function_id : integer --
negate : boolean --

View: expr_xin

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
left_operand : integer --
subquery : integer --
negate : boolean --

View: expr_xisnull

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
left_operand : integer --
negate : boolean --

View: expr_xnull

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
negate : boolean --

View: expr_xnum

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
literal : text --

View: expr_xop

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
left_operand : integer --
operator : text --
right_operand : integer --
negate : boolean --

View: expr_xser

Columns:

field name : datatype -- parameters, contraints and notes

id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
operator : text --
negate : boolean --

View: expr_xstr

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
literal : text --

View: expr_xsubq

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
parenthesize : boolean --
parent_expr : integer --
seq_no : integer --
subquery : integer --
negate : boolean --

Table: expression

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
type : text -- `NOT NULL`,
parenthesize : boolean -- `NOT NULL`, `DEFAULT false`,
parent_expr : integer -- `REFERENCES` query.expression.
seq_no : integer -- `NOT NULL`, `DEFAULT 1`,
literal : text --
table_alias : text --
column_name : text --
left_operand : integer -- `REFERENCES` query.expression.
operator : text --
right_operand : integer -- `REFERENCES` query.expression.
function_id : integer -- `REFERENCES` query.function_sig.
subquery : integer -- `REFERENCES` query.stored_query.
cast_type : integer -- `REFERENCES` query.datatype.
negate : boolean -- `NOT NULL`, `DEFAULT false`,

bind_variable : text -- REFERENCES query.bind_variable.

Constraints:

expression_type : CHECK ((type = ANY (ARRAY['xbet'::text, 'xbind'::text, 'xbool'::text, 'xcase'::text, 'xcast'::text, 'xcol'::text, 'xex'::text, 'xfunc'::text, 'xin'::text, 'xisnull'::text, 'xnull'::text, 'xnum'::text, 'xop'::text, 'xser'::text, 'xstr'::text, 'xsubq'::text])))

Tables referencing query.case_branch via Foreign Key Constraints:

query.case_branch
query.expression
query.from_relation
query.order_by_item
query.select_item
query.stored_query

Table: from_relation

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
type : text -- NOT NULL,
table_name : text --
class_name : text --
subquery : integer -- REFERENCES query.stored_query.
function_call : integer -- REFERENCES query.expression.
table_alias : text --
parent_relation : integer -- REFERENCES query.from_relation.
seq_no : integer -- NOT NULL, DEFAULT 1,
join_type : text --
on_clause : integer -- REFERENCES query.expression.

Constraints:

good_join_type : CHECK (((join_type IS NULL) OR (join_type = ANY (ARRAY['INNER'::text, 'LEFT'::text, 'RIGHT'::text, 'FULL'::text]))))
join_or_core : CHECK (((((parent_relation IS NULL) AND (join_type IS NULL)) AND (on_clause IS NULL)) OR (((parent_relation IS NOT NULL) AND (join_type IS NOT NULL)) AND (on_clause IS NOT NULL))))
relation_type : CHECK ((type = ANY (ARRAY['RELATION'::text, 'SUBQUERY'::text, 'FUNCTION'::text])))

Tables referencing query.from_relation via Foreign Key Constraints:

query.from_relation
query.record_column
query.stored_query

Table: function_param_def

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
function_id : integer -- UNIQUE#1, NOT NULL, REFERENCES query.function_sig.
seq_no : integer -- UNIQUE#1, NOT NULL,

datatype : integer -- NOT NULL, REFERENCES query.datatype.

Constraints:

qfpd_pos_seq_no : CHECK ((seq_no > 0))

Table: function_sig

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
function_name : text -- NOT NULL,
return_type : integer -- REFERENCES query.datatype.
is_aggregate : boolean -- NOT NULL, DEFAULT false,

Constraints:

qfd_rtn_or_aggr : CHECK (((return_type IS NULL) OR (is_aggregate = false)))

Indexes:

query_function_sig_name_idx : function_name

Tables referencing query.expression via Foreign Key Constraints:

query.expression query.function_param_def

Table: order_by_item

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
stored_query : integer -- UNIQUE#1, NOT NULL, REFERENCES query.stored_query.
seq_no : integer -- UNIQUE#1, NOT NULL,
expression : integer -- NOT NULL, REFERENCES query.expression.

Table: query_sequence

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
parent_query : integer -- UNIQUE#1, NOT NULL, REFERENCES query.stored_query.
seq_no : integer -- UNIQUE#1, NOT NULL,
child_query : integer -- NOT NULL, REFERENCES query.stored_query.

Table: record_column

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
from_relation : integer -- UNIQUE#1, NOT NULL, REFERENCES query.from_relation.
seq_no : integer -- UNIQUE#1, NOT NULL,
column_name : text -- NOT NULL,
column_type : integer -- NOT NULL, REFERENCES query.datatype.

Table: select_item

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
stored_query : integer -- UNIQUE#1, NOT NULL, REFERENCES query.stored_query.
seq_no : integer -- UNIQUE#1, NOT NULL,
expression : integer -- NOT NULL, REFERENCES query.expression.
column_alias : text --
grouped_by : boolean -- NOT NULL, DEFAULT false,

Table: stored_query

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
type : text -- NOT NULL,
use_all : boolean -- NOT NULL, DEFAULT false,
use_distinct : boolean -- NOT NULL, DEFAULT false,
from_clause : integer -- REFERENCES query.from_relation.
where_clause : integer -- REFERENCES query.expression.
having_clause : integer -- REFERENCES query.expression.
limit_count : integer -- REFERENCES query.expression.
offset_count : integer -- REFERENCES query.expression.

Constraints:

query_type : CHECK ((type = ANY (ARRAY['SELECT'::text, 'UNION'::text, 'INTERSECT'::text, 'EXCEPT'::text])))

Tables referencing action.fieldset via Foreign Key Constraints:

action.fieldset
query.expression
query.from_relation
query.order_by_item
query.query_sequence
query.select_item

Table: subfield

Columns:

field name : datatype -- parameters, contraints and notes

id : serial -- PRIMARY KEY,
composite_type : integer -- UNIQUE#1, NOT NULL, REFERENCES query.datatype.
seq_no : integer -- UNIQUE#1, NOT NULL,
subfield_type : integer -- NOT NULL, REFERENCES query.datatype.

Constraints:

qsf_pos_seq_no : CHECK ((seq_no > 0))

Schema reporter

View: circ_type

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
type : text --

View: currently_running

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
runner_barcode : text --
name : text --
run_time : timestamp with time zone --
scheduled_wait_time : interval --

View: demographic

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
dob : timestamp with time zone --
general_division : text --

View: hold_request_record

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
target : bigint --
hold_type : text --
bib_record : bigint --

Table: materialized_simple_record

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`,
fingerprint : text --
quality : integer --
tcn_source : text --
tcn_value : text --
title : text --
author : text --
publisher : text --
pubdate : text --
isbn : text[] --
issn : text[] --

View: old_super_simple_record

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
fingerprint : text --
quality : integer --
tcn_source : text --
tcn_value : text --
title : text --
author : text --
publisher : text --
pubdate : text --
isbn : text[] --
issn : text[] --

Table: output_folder

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
parent : integer -- `REFERENCES` reporter.output_folder.
owner : integer -- `NOT NULL`, `REFERENCES` actor.usr.
create_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
name : text -- `NOT NULL`,
shared : boolean -- `NOT NULL`, `DEFAULT false`,
share_with : integer -- `REFERENCES` actor.org_unit.

Indexes:

rpt_output_fldr_owner_idx : owner

Tables referencing reporter.output_folder via Foreign Key Constraints:

reporter.output_folder

reporter.schedule

View: overdue_circs

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
usr : integer --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
unrecovered : boolean --
target_copy : bigint --
circ_lib : integer --
circ_staff : integer --
checkin_staff : integer --
checkin_lib : integer --
renewal_remaining : integer --
due_date : timestamp with time zone --
stop_fines_time : timestamp with time zone --
checkin_time : timestamp with time zone --
create_time : timestamp with time zone --
duration : interval --
fine_interval : interval --
recurring_fine : numeric(6,2) --
max_fine : numeric(6,2) --
phone_renewal : boolean --
desk_renewal : boolean --
opac_renewal : boolean --
duration_rule : text --
recurring_fine_rule : text --
max_fine_rule : text --
stop_fines : text --
workstation : integer --
checkin_workstation : integer --
checkin_scan_time : timestamp with time zone --
parent_circ : bigint --

View: overdue_reports

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
runner_barcode : text --
name : text --
run_time : timestamp with time zone --
scheduled_wait_time : interval --

View: pending_reports

Columns:

field name : datatype -- parameters, contraints and notes
id : integer --
runner_barcode : text --
name : text --
run_time : timestamp with time zone --
scheduled_wait_time : interval --

Table: report

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
owner : integer -- `NOT NULL`, `REFERENCES` actor.usr.
create_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
name : text -- `NOT NULL`, `DEFAULT ''::text`,
description : text -- `NOT NULL`, `DEFAULT ''::text`,
template : integer -- `NOT NULL`, `REFERENCES` reporter.template.
data : text -- `NOT NULL`,
folder : integer -- `NOT NULL`, `REFERENCES` reporter.report_folder.
recur : boolean -- `NOT NULL`, `DEFAULT false`,
recurrence : interval --

Indexes:

rpt_rpt_fldr_idx : folder
rpt_rpt_owner_idx : owner

Tables referencing reporter.schedule via Foreign Key Constraints:

reporter.schedule

Table: report_folder

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
parent : integer -- `REFERENCES` reporter.report_folder.
owner : integer -- `NOT NULL`, `REFERENCES` actor.usr.
create_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
name : text -- `NOT NULL`,
shared : boolean -- `NOT NULL`, `DEFAULT false`,
share_with : integer -- `REFERENCES` actor.org_unit.

Indexes:

rpt_rpt_fldr_owner_idx : owner

Tables referencing reporter.report via Foreign Key Constraints:

reporter.report reporter.report_folder

Table: schedule

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
report : integer -- `NOT NULL`, `REFERENCES` reporter.report.
folder : integer -- `NOT NULL`, `REFERENCES` reporter.output_folder.
runner : integer -- `NOT NULL`, `REFERENCES` actor.usr.
run_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
start_time : timestamp with time zone --
complete_time : timestamp with time zone --
email : text --
excel_format : boolean -- `NOT NULL`, `DEFAULT true`,
html_format : boolean -- `NOT NULL`, `DEFAULT true`,
csv_format : boolean -- `NOT NULL`, `DEFAULT true`,
chart_pie : boolean -- `NOT NULL`, `DEFAULT false`,
chart_bar : boolean -- `NOT NULL`, `DEFAULT false`,
chart_line : boolean -- `NOT NULL`, `DEFAULT false`,
error_code : integer --
error_text : text --

Indexes:

rpt_sched_folder_idx : folder
rpt_sched_runner_idx : runner

View: simple_record

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
metarecord : bigint --
fingerprint : text --
quality : integer --
tcn_source : text --
tcn_value : text --
title : text --
uniform_title : text --
author : text --
publisher : text --
pubdate : text --
series_title : text --
series_statement : text --
summary : text --
isbn : text[] --

issn : text[] --
topic_subject : text[] --
geographic_subject : text[] --
genre : text[] --
name_subject : text[] --
corporate_subject : text[] --
external_uri : text[] --

View: super_simple_record

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
fingerprint : text --
quality : integer --
tcn_source : text --
tcn_value : text --
title : text --
author : text --
publisher : text --
pubdate : text --
isbn : text[] --
issn : text[] --

Table: template

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
owner : integer -- `NOT NULL`, `REFERENCES` actor.usr.
create_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
name : text -- `NOT NULL`,
description : text -- `NOT NULL`,
data : text -- `NOT NULL`,
folder : integer -- `NOT NULL`, `REFERENCES` reporter.template_folder.

Indexes:

rpt_tmpl_fldr_idx : folder
rpt_tmpl_owner_idx : owner

Tables referencing reporter.report via Foreign Key Constraints:

reporter.report

Table: template_folder

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
parent : integer -- `REFERENCES` reporter.template_folder.
owner : integer -- `NOT NULL`, `REFERENCES` actor.usr.
create_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
name : text -- `NOT NULL`,
shared : boolean -- `NOT NULL`, `DEFAULT false`,
share_with : integer -- `REFERENCES` actor.org_unit.

Indexes:

rpt_tmpl_fldr_owner_idx : owner

Tables referencing reporter.template via Foreign Key Constraints:

reporter.template

reporter.template_folder

View: xact_billing_totals

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
unvoided : numeric --
voided : numeric --
total : numeric --

View: xact_paid_totals

Columns:

field name : datatype -- parameters, contraints and notes
xact : bigint --
unvoided : numeric --
voided : numeric --
total : numeric --

disable_materialized_simple_record_trigger()

Function Properties

Language: SQL
Return Type: void

enable_materialized_simple_record_trigger()

Function Properties

Language: SQL
Return Type: void

refresh_materialized_simple_record()

Function Properties

Language: SQL
Return Type: void

simple_rec_delete(r_id bigint)

Function Properties

Language: SQL
Return Type: boolean

simple_rec_trigger()

Function Properties

Language: PLPGSQL
Return Type: trigger

simple_rec_update(deleted bigint, r_id boolean)

Function Properties

Language: PLPGSQL
Return Type: boolean

simple_rec_update(r_id bigint)

Function Properties

Language: SQL
Return Type: boolean

Schema search

Table: relevance_adjustment

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
active : boolean -- `NOT NULL`, `DEFAULT true`,
field : integer -- `NOT NULL`, `REFERENCES` config.metabib_field.
bump_type : text -- `NOT NULL`,
multiplier : numeric -- `NOT NULL`, `DEFAULT 1.0`,

Constraints:

relevance_adjustment_bump_type_check : CHECK ((bump_type = ANY (ARRAY['word_order'::text, 'first_word'::text, 'full_match'::text])))

explode_array(anyarray)

Function Properties

Language: SQL
Return Type: SET OF anyelement

query_parser_fts(staff integer, metarecord integer, param_limit text, param_check integer[], param_offset integer[], param_locations integer, param_statuses integer, param_query integer, param_depth boolean, param_search_ou boolean)

Function Properties

Language: PLPGSQL
Return Type: SET OF search_result

Schema serial

Table: basic_summary

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
distribution : integer -- `NOT NULL`, `REFERENCES` serial.distribution.
generated_coverage : text -- `NOT NULL`,
textual_holdings : text --
show_generated : boolean -- `NOT NULL`, `DEFAULT true`,

Indexes:

serial_basic_summary_dist_idx : distribution

Table: caption_and_pattern

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
subscription : integer -- `NOT NULL`, `REFERENCES` serial.subscription.

type : text -- NOT NULL,
create_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
start_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
end_date : timestamp with time zone --
active : boolean -- NOT NULL, DEFAULT false,
pattern_code : text -- NOT NULL,
enum_1 : text --
enum_2 : text --
enum_3 : text --
enum_4 : text --
enum_5 : text --
enum_6 : text --
chron_1 : text --
chron_2 : text --
chron_3 : text --
chron_4 : text --
chron_5 : text --

Constraints:

cap_type : CHECK ((type = ANY (ARRAY['basic'::text, 'supplement'::text, 'index'::text])))

Indexes:

serial_caption_and_pattern_sub_idx : subscription

Tables referencing serial.issuance via Foreign Key Constraints:

serial.issuance

Table: distribution

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
record_entry : bigint -- REFERENCES serial.record_entry.
summary_method : text --
subscription : integer -- NOT NULL, REFERENCES serial.subscription.
holding_lib : integer -- NOT NULL, REFERENCES actor.org_unit.
label : text -- NOT NULL,
receive_call_number : bigint -- REFERENCES asset.call_number.
receive_unit_template : integer -- REFERENCES asset.copy_template.
bind_call_number : bigint -- REFERENCES asset.call_number.
bind_unit_template : integer -- REFERENCES asset.copy_template.
unit_label_prefix : text --
unit_label_suffix : text --

Constraints:

sdist_summary_method_check : CHECK (((summary_method IS NULL) OR (summary_method = ANY (ARRAY['add_to_sre'::text, 'merge_with_sre'::text, 'use_sre_only'::text, 'use_sdist_only'::text]))))

Indexes:

serial_distribution_holding_lib_idx : holding_lib
serial_distribution_sub_idx : subscription

Tables referencing serial.basic_summary via Foreign Key Constraints:

serial.basic_summary
serial.distribution_note
serial.index_summary
serial.stream
serial.supplement_summary

Table: distribution_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
distribution : integer -- `NOT NULL`, `REFERENCES` serial.distribution.
creator : integer -- `NOT NULL`, `REFERENCES` actor.usr.
create_date : timestamp with time zone -- `DEFAULT now()`,
pub : boolean -- `NOT NULL`, `DEFAULT false`,
title : text -- `NOT NULL`,
value : text -- `NOT NULL`,

Indexes:

serial_distribution_note_dist_idx : distribution

Table: index_summary

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
distribution : integer -- `NOT NULL`, `REFERENCES` serial.distribution.
generated_coverage : text -- `NOT NULL`,
textual_holdings : text --
show_generated : boolean -- `NOT NULL`, `DEFAULT true`,

Indexes:

serial_index_summary_dist_idx : distribution

Table: issuance

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
creator : integer -- `NOT NULL`, `REFERENCES` actor.usr.
editor : integer -- `NOT NULL`, `REFERENCES` actor.usr.
create_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,

edit_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
subscription : integer -- NOT NULL, REFERENCES serial.subscription.
label : text --
date_published : timestamp with time zone --
caption_and_pattern : integer -- REFERENCES serial.caption_and_pattern.
holding_code : text --
holding_type : text --
holding_link_id : integer --

Constraints:

valid_holding_type : CHECK (((holding_type IS NULL) OR (holding_type = ANY (ARRAY['basic'::text, 'supplement'::text, 'index'::text]))))

Indexes:

serial_issuance_caption_and_pattern_idx : caption_and_pattern
serial_issuance_date_published_idx : date_published
serial_issuance_sub_idx : subscription

Tables referencing serial.item via Foreign Key Constraints:

serial.item

Table: item

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
creator : integer -- NOT NULL, REFERENCES actor.usr.
editor : integer -- NOT NULL, REFERENCES actor.usr.
create_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
edit_date : timestamp with time zone -- NOT NULL, DEFAULT now(),
issuance : integer -- NOT NULL, REFERENCES serial.issuance.
stream : integer -- NOT NULL, REFERENCES serial.stream.
unit : integer -- REFERENCES serial.unit.
uri : integer -- REFERENCES asset.uri.
date_expected : timestamp with time zone --
date_received : timestamp with time zone --
status : text -- DEFAULT 'Expected'::text,
shadowed : boolean -- NOT NULL, DEFAULT false,

Constraints:

valid_status : CHECK ((status = ANY (ARRAY['Bindery'::text, 'Bound'::text, 'Claimed'::text, 'Discarded'::text, 'Expected'::text, 'Not Held'::text, 'Not Published'::text, 'Received'::text])))

Indexes:

serial_item_date_received_idx : date_received
serial_item_issuance_idx : issuance
serial_item_status_idx : status

serial_item_stream_idx : stream
serial_item_unit_idx : unit
serial_item_uri_idx : uri

Tables referencing acq.serial_claim via Foreign Key Constraints:

acq.serial_claim

serial.item_note

Table: item_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
item : integer -- `NOT NULL`, `REFERENCES` serial.item.
creator : integer -- `NOT NULL`, `REFERENCES` actor.usr.
create_date : timestamp with time zone -- `DEFAULT now()`,
pub : boolean -- `NOT NULL`, `DEFAULT false`,
title : text -- `NOT NULL`,
value : text -- `NOT NULL`,

Indexes:

serial_item_note_item_idx : item

Table: record_entry

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
record : bigint -- `REFERENCES` biblio.record_entry.
owning_lib : integer -- `NOT NULL`, `DEFAULT 1`, `REFERENCES` actor.org_unit.
creator : integer -- `NOT NULL`, `DEFAULT 1`,
editor : integer -- `NOT NULL`, `DEFAULT 1`,
source : integer --
create_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
edit_date : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
active : boolean -- `NOT NULL`, `DEFAULT true`,
deleted : boolean -- `NOT NULL`, `DEFAULT false`,
marc : text --
last_xact_id : text -- `NOT NULL`,

Indexes:

serial_record_entry_creator_idx : creator
serial_record_entry_editor_idx : editor
serial_record_entry_owning_lib_idx : owning_lib, deleted

Tables referencing serial.distribution via Foreign Key Constraints:

serial.distribution

Table: routing_list_user

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
stream : integer -- UNIQUE#1, NOT NULL, REFERENCES serial.stream.
pos : integer -- UNIQUE#1, NOT NULL, DEFAULT 1,
reader : integer -- REFERENCES actor.usr.
department : text --
note : text --

Constraints:

reader_or_dept : CHECK ((((reader IS NOT NULL) AND (department IS NULL)) OR ((reader IS NULL) AND (department IS NOT NULL))))

Indexes:

serial_routing_list_user_reader_idx : reader
serial_routing_list_user_stream_idx : stream

Table: stream

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
distribution : integer -- NOT NULL, REFERENCES serial.distribution.
routing_label : text --

Indexes:

serial_stream_dist_idx : distribution

Tables referencing serial.item via Foreign Key Constraints:

serial.item serial.routing_list_user

Table: subscription

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
owning_lib : integer -- NOT NULL, DEFAULT 1, REFERENCES actor.org_unit.
start_date : timestamp with time zone -- NOT NULL,
end_date : timestamp with time zone --
record_entry : bigint -- REFERENCES biblio.record_entry.
expected_date_offset : interval --

Indexes:

serial_subscription_owner_idx : owning_lib
serial_subscription_record_idx : record_entry

Tables referencing serial.caption_and_pattern via Foreign Key Constraints:

serial.caption_and_pattern
serial.distribution
serial.issuance
serial.subscription_note

Table: subscription_note

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
subscription : integer -- `NOT NULL`, `REFERENCES` serial.subscription.
creator : integer -- `NOT NULL`, `REFERENCES` actor.usr.
create_date : timestamp with time zone -- `DEFAULT now()`,
pub : boolean -- `NOT NULL`, `DEFAULT false`,
title : text -- `NOT NULL`,
value : text -- `NOT NULL`,

Indexes:

serial_subscription_note_sub_idx : subscription

Table: supplement_summary

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- `PRIMARY KEY`,
distribution : integer -- `NOT NULL`, `REFERENCES` serial.distribution.
generated_coverage : text -- `NOT NULL`,
textual_holdings : text --
show_generated : boolean -- `NOT NULL`, `DEFAULT true`,

Indexes:

serial_supplement_summary_dist_idx : distribution

Table: unit

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- `PRIMARY KEY`, `DEFAULT nextval('asset.copy_id_seq'::regclass)`,
circ_lib : integer -- `NOT NULL`,
creator : bigint -- `NOT NULL`, `REFERENCES` actor.usr.
call_number : bigint -- `NOT NULL`, `REFERENCES` asset.call_number.
editor : bigint -- `NOT NULL`, `REFERENCES` actor.usr.
create_date : timestamp with time zone -- `DEFAULT now()`,
edit_date : timestamp with time zone -- `DEFAULT now()`,

copy_number : integer --
status : integer -- NOT NULL,
location : integer -- NOT NULL, DEFAULT 1,
loan_duration : integer -- NOT NULL,
fine_level : integer -- NOT NULL,
age_protect : integer --
circulate : boolean -- NOT NULL, DEFAULT true,
deposit : boolean -- NOT NULL, DEFAULT false,
ref : boolean -- NOT NULL, DEFAULT false,
holdable : boolean -- NOT NULL, DEFAULT true,
deposit_amount : numeric(6,2) -- NOT NULL, DEFAULT 0.00,
price : numeric(8,2) --
barcode : text -- NOT NULL,
circ_modifier : text --
circ_as_type : text --
dummy_title : text --
dummy_author : text --
alert_message : text --
opac_visible : boolean -- NOT NULL, DEFAULT true,
deleted : boolean -- NOT NULL, DEFAULT false,
floating : boolean -- NOT NULL, DEFAULT false,
dummy_isbn : text --
status_changed_time : timestamp with time zone --
mint_condition : boolean -- NOT NULL, DEFAULT true,
cost : numeric(8,2) --
sort_key : text --
detailed_contents : text -- NOT NULL,
summary_contents : text -- NOT NULL,

Constraints:

copy_fine_level_check : CHECK ((fine_level = ANY (ARRAY[1, 2, 3])))
copy_loan_duration_check : CHECK ((loan_duration = ANY (ARRAY[1, 2, 3])))

Indexes:

unit_avail_cn_idx : call_number
unit_cn_idx : call_number
unit_creator_idx : creator
unit_editor_idx : editor

Tables referencing serial.item via Foreign Key Constraints:

serial.item

Schema staging

Table: billing_address_stage

Columns:

field name : datatype -- parameters, contraints and notes
row_id : bigint -- PRIMARY KEY, DEFAULT nextval('staging.mailing_address_stage_row_id_seq'::regclass),
row_date : timestamp with time zone -- DEFAULT now(),
usrname : text -- NOT NULL,
street1 : text --
street2 : text --
city : text -- NOT NULL, DEFAULT ''::text,
state : text -- NOT NULL, DEFAULT 'OK'::text,
country : text -- NOT NULL, DEFAULT 'US'::text,
post_code : text -- NOT NULL,
complete : boolean -- DEFAULT false,

Table: card_stage

Columns:

field name : datatype -- parameters, contraints and notes
row_id : bigserial -- PRIMARY KEY,
row_date : timestamp with time zone -- DEFAULT now(),
usrname : text -- NOT NULL,
barcode : text -- NOT NULL,
complete : boolean -- DEFAULT false,

Table: mailing_address_stage

Columns:

field name : datatype -- parameters, contraints and notes
row_id : bigserial -- PRIMARY KEY,
row_date : timestamp with time zone -- DEFAULT now(),
usrname : text -- NOT NULL,
street1 : text --
street2 : text --
city : text -- NOT NULL, DEFAULT ''::text,
state : text -- NOT NULL, DEFAULT 'OK'::text,
country : text -- NOT NULL, DEFAULT 'US'::text,
post_code : text -- NOT NULL,
complete : boolean -- DEFAULT false,

Table: statcat_stage

Columns:

field name : datatype -- parameters, contraints and notes
row_id : bigserial -- PRIMARY KEY,
row_date : timestamp with time zone -- DEFAULT now(),
usrname : text -- NOT NULL,
statcat : text -- NOT NULL,
value : text -- NOT NULL,

complete : boolean -- DEFAULT false,

Table: user_stage

Columns:

field name : datatype -- parameters, contraints and notes
row_id : bigserial -- PRIMARY KEY,
row_date : timestamp with time zone -- DEFAULT now(),
usrname : text -- NOT NULL,
profile : text --
email : text --
passwd : text --
ident_type : integer -- DEFAULT 3,
first_given_name : text --
second_given_name : text --
family_name : text --
day_phone : text --
evening_phone : text --
home_ou : integer -- DEFAULT 2,
dob : text --
complete : boolean -- DEFAULT false,

Schema stats

View: fleshed_call_number

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
creator : bigint --
create_date : timestamp with time zone --
editor : bigint --
edit_date : timestamp with time zone --
record : bigint --
owning_lib : integer --
label : text --
deleted : boolean --
label_class : bigint --
label_sortkey : text --
create_date_day : date --
edit_date_day : date --
create_date_hour : timestamp with time zone --
edit_date_hour : timestamp with time zone --
item_lang : text --
item_type : text --
item_form : text --

View: fleshed_circulation

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint --
usr : integer --
xact_start : timestamp with time zone --
xact_finish : timestamp with time zone --
unrecovered : boolean --
target_copy : bigint --
circ_lib : integer --
circ_staff : integer --
checkin_staff : integer --
checkin_lib : integer --
renewal_remaining : integer --
due_date : timestamp with time zone --
stop_fines_time : timestamp with time zone --
checkin_time : timestamp with time zone --
create_time : timestamp with time zone --
duration : interval --
fine_interval : interval --
recurring_fine : numeric(6,2) --
max_fine : numeric(6,2) --
phone_renewal : boolean --
desk_renewal : boolean --
opac_renewal : boolean --
duration_rule : text --
recurring_fine_rule : text --
max_fine_rule : text --
stop_fines : text --
workstation : integer --
checkin_workstation : integer --
checkin_scan_time : timestamp with time zone --
parent_circ : bigint --
start_date_day : date --
finish_date_day : date --
start_date_hour : timestamp with time zone --
finish_date_hour : timestamp with time zone --
call_number_label : text --
owning_lib : integer --
item_lang : text --
item_type : text --
item_form : text --

View: fleshed_copy

Columns:

field name : datatype -- parameters, contraints and notes

id : bigint --
circ_lib : integer --
creator : bigint --
call_number : bigint --
editor : bigint --
create_date : timestamp with time zone --
edit_date : timestamp with time zone --
copy_number : integer --
status : integer --
location : integer --
loan_duration : integer --
fine_level : integer --
age_protect : integer --
circulate : boolean --
deposit : boolean --
ref : boolean --
holdable : boolean --
deposit_amount : numeric(6,2) --
price : numeric(8,2) --
barcode : text --
circ_modifier : text --
circ_as_type : text --
dummy_title : text --
dummy_author : text --
alert_message : text --
opac_visible : boolean --
deleted : boolean --
floating : boolean --
dummy_isbn : text --
status_changed_time : timestamp with time zone --
mint_condition : boolean --
cost : numeric(8,2) --
create_date_day : date --
edit_date_day : date --
create_date_hour : timestamp with time zone --
edit_date_hour : timestamp with time zone --
call_number_label : text --
owning_lib : integer --
item_lang : text --
item_type : text --
item_form : text --

Schema vandelay

Table: authority_attr_definition

Columns:

field name : datatype -- parameters, contraints and notes

id : serial -- PRIMARY KEY,
code : text -- UNIQUE, NOT NULL,
description : text --
xpath : text -- NOT NULL,
remove : text -- NOT NULL, DEFAULT ''::text,
ident : boolean -- NOT NULL, DEFAULT false,

Tables referencing vandelay.queued_authority_record_attr via Foreign Key Constraints:

vandelay.queued_authority_record_attr

Table: authority_match

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
matched_attr : integer -- REFERENCES vandelay.queued_authority_record_attr.
queued_record : bigint -- REFERENCES vandelay.queued_authority_record.
eg_record : bigint -- REFERENCES authority.record_entry.

Table: authority_queue

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('vandelay.queue_id_seq'::regclass),
owner : integer -- UNIQUE#1, NOT NULL,
name : text -- UNIQUE#1, NOT NULL,
complete : boolean -- NOT NULL, DEFAULT false,
queue_type : text -- UNIQUE#1, NOT NULL, DEFAULT 'authority'::text,

Constraints:

authority_queue_queue_type_check : CHECK ((queue_type = 'authority'::text))
queue_queue_type_check : CHECK ((queue_type = ANY (ARRAY['bib'::text, 'authority'::text])))

Tables referencing vandelay.queued_authority_record via Foreign Key Constraints:

vandelay.queued_authority_record

Table: bib_attr_definition

Columns:

field name : datatype -- parameters, contraints and notes
id : serial -- PRIMARY KEY,
code : text -- UNIQUE, NOT NULL,
description : text --
xpath : text -- NOT NULL,

remove : text -- NOT NULL, DEFAULT ''::text,
ident : boolean -- NOT NULL, DEFAULT false,

Tables referencing vandelay.queued_bib_record_attr via Foreign Key Constraints:

vandelay.queued_bib_record_attr

Table: bib_match

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
field_type : text -- NOT NULL,
matched_attr : integer -- REFERENCES vandelay.queued_bib_record_attr.
queued_record : bigint -- REFERENCES vandelay.queued_bib_record.
eg_record : bigint -- REFERENCES biblio.record_entry.

Constraints:

bib_match_field_type_check : CHECK ((field_type = ANY (ARRAY['isbn'::text, 'tcn_value'::text, 'id'::text])))

Table: bib_queue

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('vandelay.queue_id_seq'::regclass),
owner : integer -- UNIQUE#1, NOT NULL,
name : text -- UNIQUE#1, NOT NULL,
complete : boolean -- NOT NULL, DEFAULT false,
queue_type : text -- UNIQUE#1, NOT NULL, DEFAULT 'bib'::text,
item_attr_def : bigint -- REFERENCES vandelay.import_item_attr_definition.

Constraints:

bib_queue_queue_type_check : CHECK ((queue_type = 'bib'::text))
queue_queue_type_check : CHECK ((queue_type = ANY (ARRAY['bib'::text, 'authority'::text])))

Tables referencing vandelay.queued_bib_record via Foreign Key Constraints:

vandelay.queued_bib_record

Table: import_bib_trash_fields

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
owner : integer -- UNIQUE#1, NOT NULL, REFERENCES actor.org_unit.
field : text -- UNIQUE#1, NOT NULL,

Table: import_item

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
record : bigint -- `NOT NULL`, `REFERENCES` vandelay.queued_bib_record.
definition : bigint -- `NOT NULL`, `REFERENCES` vandelay.import_item_attr_definition.
owning_lib : integer --
circ_lib : integer --
call_number : text --
copy_number : integer --
status : integer --
location : integer --
circulate : boolean --
deposit : boolean --
deposit_amount : numeric(8,2) --
ref : boolean --
holdable : boolean --
price : numeric(8,2) --
barcode : text --
circ_modifier : text --
circ_as_type : text --
alert_message : text --
pub_note : text --
priv_note : text --
opac_visible : boolean --

Table: import_item_attr_definition

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
owner : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.org_unit.
name : text -- `UNIQUE#1`, `NOT NULL`,
tag : text -- `NOT NULL`,
keep : boolean -- `NOT NULL`, `DEFAULT false`,
owning_lib : text --
circ_lib : text --
call_number : text --
copy_number : text --
status : text --
location : text --
circulate : text --
deposit : text --
deposit_amount : text --
ref : text --
holdable : text --
price : text --

barcode : text --
circ_modifier : text --
circ_as_type : text --
alert_message : text --
opac_visible : text --
pub_note_title : text --
pub_note : text --
priv_note_title : text --
priv_note : text --

Tables referencing vandelay.bib_queue via Foreign Key Constraints:

vandelay.bib_queue vandelay.import_item

Table: merge_profile

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
owner : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.org_unit.
name : text -- `UNIQUE#1`, `NOT NULL`,
add_spec : text --
replace_spec : text --
strip_spec : text --
preserve_spec : text --

Constraints:

add_replace_strip_or_preserve : CHECK ((((preserve_spec IS NOT NULL) OR (replace_spec IS NOT NULL)) OR ((preserve_spec IS NULL) AND (replace_spec IS NULL))))

Table: queue

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
owner : integer -- `UNIQUE#1`, `NOT NULL`, `REFERENCES` actor.usr.
name : text -- `UNIQUE#1`, `NOT NULL`,
complete : boolean -- `NOT NULL`, `DEFAULT false`,
queue_type : text -- `UNIQUE#1`, `NOT NULL`, `DEFAULT 'bib'::text`,

Constraints:

queue_queue_type_check : CHECK ((queue_type = ANY (ARRAY['bib'::text, 'authority'::text])))

Table: queued_authority_record

Columns:

field name : datatype -- parameters, contraints and notes

id : bigint -- PRIMARY KEY, DEFAULT nextval('vandelay.queued_record_id_seq'::regclass),
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
import_time : timestamp with time zone --
purpose : text -- NOT NULL, DEFAULT 'import'::text,
marc : text -- NOT NULL,
queue : integer -- NOT NULL, REFERENCES vandelay.authority_queue.
imported_as : integer -- REFERENCES authority.record_entry.

Constraints:

queued_record_purpose_check : CHECK ((purpose = ANY (ARRAY['import'::text, 'overlay'::text])))

Indexes:

queued_authority_record_queue_idx : queue

Tables referencing vandelay.authority_match via Foreign Key Constraints:

vandelay.authority_match vandelay.queued_authority_record_attr

Table: queued_authority_record_attr

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- PRIMARY KEY,
record : bigint -- NOT NULL, REFERENCES vandelay.queued_authority_record.
field : integer -- NOT NULL, REFERENCES vandelay.authority_attr_definition.
attr_value : text -- NOT NULL,

Indexes:

queued_authority_record_attr_record_idx : record

Tables referencing vandelay.authority_match via Foreign Key Constraints:

vandelay.authority_match

Table: queued_bib_record

Columns:

field name : datatype -- parameters, contraints and notes
id : bigint -- PRIMARY KEY, DEFAULT nextval('vandelay.queued_record_id_seq'::regclass),
create_time : timestamp with time zone -- NOT NULL, DEFAULT now(),
import_time : timestamp with time zone --
purpose : text -- NOT NULL, DEFAULT 'import'::text,
marc : text -- NOT NULL,
queue : integer -- NOT NULL, REFERENCES vandelay.bib_queue.
bib_source : integer -- REFERENCES config.bib_source.
imported_as : bigint -- REFERENCES biblio.record_entry.

Constraints:

queued_record_purpose_check : CHECK ((purpose = ANY (ARRAY['import'::text, 'overlay'::text])))

Indexes:

queued_bib_record_queue_idx : queue

Tables referencing vandelay.bib_match via Foreign Key Constraints:

vandelay.bib_match vandelay.import_item
vandelay.queued_bib_record_attr

Table: queued_bib_record_attr

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
record : bigint -- `NOT NULL`, `REFERENCES` vandelay.queued_bib_record.
field : integer -- `NOT NULL`, `REFERENCES` vandelay.bib_attr_definition.
attr_value : text -- `NOT NULL`,

Indexes:

queued_bib_record_attr_record_idx : record

Tables referencing vandelay.bib_match via Foreign Key Constraints:

vandelay.bib_match

Table: queued_record

Columns:

field name : datatype -- parameters, contraints and notes
id : bigserial -- `PRIMARY KEY`,
create_time : timestamp with time zone -- `NOT NULL`, `DEFAULT now()`,
import_time : timestamp with time zone --
purpose : text -- `NOT NULL`, `DEFAULT 'import'::text`,
marc : text -- `NOT NULL`,

Constraints:

queued_record_purpose_check : CHECK ((purpose = ANY (ARRAY['import'::text, 'overlay'::text])))

add_field(field text, source_xml text, target_xml text)

Function Properties

Language: SQL
Return Type: text

add_field(force_add text, field text, source_xml text, target_xml integer)

Function Properties

Language: PLPERLU
Return Type: text

auto_overlay_authority_queue(merge_profile_id bigint, queue_id integer)

Function Properties

Language: PLPGSQL
Return Type: SET OF bigint

auto_overlay_authority_queue(queue_id bigint)

Function Properties

Language: SQL
Return Type: SET OF bigint

auto_overlay_authority_record(merge_profile_id bigint, import_id integer)

Function Properties

Language: PLPGSQL
Return Type: boolean

auto_overlay_bib_queue(merge_profile_id bigint, queue_id integer)

Function Properties

Language: PLPGSQL
Return Type: SET OF bigint

auto_overlay_bib_queue(queue_id bigint)

Function Properties

Language: SQL
Return Type: SET OF bigint

auto_overlay_bib_record(merge_profile_id bigint, import_id integer)

Function Properties

Language: PLPGSQL
Return Type: boolean

cleanup_authority_marc()

Function Properties

Language: PLPGSQL
Return Type: trigger

cleanup_bib_marc()

Function Properties

Language: PLPGSQL
Return Type: trigger

compile_profile(incoming_xml text)

Function Properties

Language: PLPGSQL
Return Type: compile_profile

find_bib_tcn_data(xml text)

Function Properties

Language: PLPGSQL
Return Type: SET OF tcn_data

ingest_authority_marc()

Function Properties

Language: PLPGSQL
Return Type: trigger

ingest_bib_items()

Function Properties

Language: PLPGSQL

Return Type: trigger

ingest_bib_marc()

Function Properties

Language: PLPGSQL
Return Type: trigger

ingest_items(attr_def_id bigint, import_id bigint)

Function Properties

Language: PLPGSQL
Return Type: SET OF import_item

match_bib_record()

Function Properties

Language: PLPGSQL
Return Type: trigger

merge_record_xml(strip_rule text, replace_preserve_rule text, add_rule text, source_xml text, target_xml text)

Function Properties

Language: SQL
Return Type: text

merge_record_xml(template_marc text, target_marc text)

Function Properties

Language: PLPGSQL
Return Type: text

overlay_authority_record(merge_profile_id bigint, eg_id bigint, import_id integer)

Function Properties

Language: PLPGSQL

Return Type: boolean

overlay_bib_record(merge_profile_id bigint, eg_id bigint, import_id integer)

Function Properties

Language: PLPGSQL
Return Type: boolean

replace_field(field text, source_xml text, target_xml text)

Function Properties

Language: PLPGSQL
Return Type: text

strip_field(field text, xml text)

Function Properties

Language: PLPERLU
Return Type: text

template_overlay_bib_record(eg_id text, v_marc bigint)

Function Properties

Language: SQL
Return Type: boolean

template_overlay_bib_record(merge_profile_id text, eg_id bigint, v_marc integer)

Function Properties

Language: PLPGSQL
Return Type: boolean

Appendix B. About this Documentation

About the Documentation Interest Group (DIG)

The Evergreen DIG was established in May 2009 at the first Evergreen International Conference, where members of the Evergreen community committed to developing single-source, standards-based documentation for Evergreen. Since then, the DIG has been actively working toward that goal.

Table B.1. Evergreen DIG Participants

Name	Organization
Hilary Caws-Elwitt	Susquehanna County Library
Karen Collier	Kent County Public Library
Shannon Dineen	SITKA
George Duimovich	NRCan Library
Sally Fortin	Equinox Software
Wolf Halton	Lysaris
Tina Ji	SITKA
Catherine Lemmer	Indiana State Library
June Rayner	eiNetwork
Tara Robertson	N/A
Rod Schiffman	Alpha-G Consulting
Steve Sheppard	Open
Ben Shum	Bibliomation
Roni Shwaish	eiNetwork
Robert Soulliere	Mohawk College
Tim Spindler	C/W MARS
Lindsay Stratton	Pioneer Library System
Yamil Suarez	Berklee College of Music
Jenny Turner	PALS
Repke de Vries	International Institute for Social History
Tigran Zargaryan	Fundamental Scientific Library of the National Academy of Sciences

Attributions

How to Participate

Contributing to documentation is an excellent way to support Evergreen, even if you are new to documentation. In fact, beginners often have a distinct advantage over the experts, more easily spotting the places where documentation is lacking or where it is unclear.

We welcome your contribution with planning, writing, editing, testing, translating to DocBook, and other tasks. Whatever your background or experience we are keen to have your help!

What you can do:

- Join the Evergreen documentation listserv: list.georgialibraries.org/mailman/listinfo/open-ils-documentation . This is the primary way we communicate with each other. Please send an email introducing yourself to the list.
- Add yourself to the participant list if you have an Evergreen DokuWiki account, or send a request to <docs@evergreen-ils.org>.
- Check out the documentation outline to see which areas need work, and let the DIG list know in which areas you would like to work.
- Review the documentation and report any error or make suggestion using Launchpad.

Volunteer Roles

We are now looking for people to help produce the documentation. If you interested in participating, email the DIG facilitators at <docs@evergreen-ils.org> or post on the documentation mailing list. We're looking for volunteers to work on the following:

- Writing – Produce the documentation (“from scratch,” and/or revised from existing materials). We're open to receiving content in any formats, such as Word or Open Office, but of course, would be most delighted with DocBook xml format.
- Testing – Compare the documents with the functions they describe and ensuring that the procedures accomplish the desired results. Even if you are not officially in the DIG, we would appreciate any suggestions you may have for Evergreen documentation.

- XML conversion – Convert existing documentation to DocBook format.
- Editorial review – Ensuring the documentation is clear and follows Evergreen DIG style guide conventions.
- Style and Design – Edit the DocBook style sheets or post style tips and suggestions on the DIG list.

Appendix C. Getting More Information

This documentation is just one way to learn about Evergreen and find solutions to Evergreen challenges. Below is a list of many other resources to help you find answers to almost any question you might have.

Evergreen Wiki - Loads of information and the main portal to the Evergreen community.

Evergreen mailing lists - These are excellent for initiating questions. There are several lists including:

- General list - General inquiries regarding Evergreen. If unsure about which list to use, this is a good stating point.
- Developer list - Technical questions should be asked here including questions regarding installation. As well, patches can be submitted using this list and developer communication also takes place here.
- DIG list - This list is used for questions and feedback regarding this documentation, the Documentation Interest Group and other documentation related ideas and issues.

Evergreen Blog - Great for getting general news and updates about Evergreen. It is also an interesting historical read with entries dating back to the early beginnings of Evergreen.

Evergreen IRC channel - Allows live chat. Many developers hang out here and will try to field technical questions. This is often the quickest way to get a solution to a specific problem. Just remember that while the channel is open 24/7, there are times when no one is available in the channel. The most active times for the IRC channel seem to be weekday afternoons (Eastern Standard Time). There is also an archive of logs from the chat sessions available on the IRC page.

Evergreen related community blogs - Evergreen related blog entries from the community.

Resource Sharing Cooperative of Evergreen Libraries (RSCEL) - Provides some technical documents and a means for the Evergreen community to collaborate with other libraries.

List of current Evergreen libraries - Locate other libraries who are using Evergreen.

Glossary

In this section we expand acronyms, define terms, and generally try to explain concepts used by Evergreen software.

A

Apache
: Open-source web server software used to serve both static content and dynamic web pages in a secure and reliable way. More information is available at http://apache.org.

B

Bookbags
: Bookbags are lists of items that can be used for any number of purposes. For example, to keep track of what books you have read, books you would like to read, to maintain a class reading list, to maintain a reading list for a book club, to keep a list of books you would like for your birthday. There are an unlimited number of uses.

C

CentOS
: A popular open-source operating system based on Red Hat Enterprises `Linux` (also known as "RHEL") and often used for in web servers. More information is available at http://www.centos.org.

Closure Compiler
: A suite of open-source tools used to build web applications with Javascript; originally developed by Google. It is used to create special builds of the Evergreen Staff Client. More information is available at http://code.google.com/closure/compiler/.

CPAN
: An open-source archive of software modules written in Perl. More information is available at http://www.cpan.org.
See Also Perl.

D

Debian
: One of the most popular open-source operating system based on the `Linux` kernel that provides over 25000 useful precompiled software packages. Also known as `Debian GNU/Linux`. More information is available at http://www.debian.org.
See Also Fedora, Ubuntu.

Domain name
: A unique set of case-insensitive, alphanumeric strings separated by periods that are used to name organizations, web sites and addresses on the Internet (e.g.: `www.esilibrary.com`). Domain names can be reserved via third-party registration services, and can be associated with a unique IP address or suite of IP addresses.
See Also IP Address.

E

ejabberd

An open-source Jabber/XMPP instant messaging server that is used for client-server message passing within Evergreen. It runs under popular operating systems (e.g., `Mac OSX`, `GNU/Linux`, and `Microsoft Windows`). One popular use is to provide XMPP messaging services for a Jabber domain across an extendable cluster of cheap, easily-replaced machine nodes. More information is available at http://www.ejabberd.im.
See Also Jabber, XMPP.

F

Fedora

A popular open-source operating system based on the `Linux` kernel. More information is available at http://fedoraproject.org/.
See Also Debian, Ubuntu.

G

Gentoo

A popular open-source operating system built on the `Linux` kernel. More information is available at http://www.gentoo.org.

H

I

IP Address

(Internet Protocol address) A numerical label consisting of four numbers separated by periods (e.g., "192.168.1.15") assigned to individual members of networked computing systems. It uniquely identifies each system on the network and allows controlled communication between such systems. The numerical label scheme must adhere to a strictly defined naming convention that is currently defined and overseen by the Internet Corporation for Assigned Names and Numbers ("ICANN").

Item/copy Buckets

Virtual “containers” to use in batch processing of item or copy records. They can be used to perform various cataloging/holdings maintenance tasks in batch.

J

Jabber

The communications protocol used for client-server message passing within Evergreen. Now known as XMPP (eXtensible Messaging and Presence Protocol), it was originally named "Jabber".
See Also XMPP, ejabberd.

K

L

M

MARC
: The MARC formats are standards for the representation and communication of bibliographic and related information in machine-readable form.

MARCXML
: Framework for working with MARC data in a XML environment.

McCoy
: An open-source application that allows add-on authors to provide secure updates to their users. It is used to create special builds of the Evergreen Staff Client. More information is available at http://developer.mozilla.org/en/McCoy.

memcached
: A general-purpose distributed memory caching system, usually with a client-server architecture spread over multiple computing systems. It reduces the number of times a data source (e.g., a database) must be directly accessed by temporarily caching data in memory, therefore dramatically speeding up database-driven web applications.

N

Network address
: Also known as an IP address (Internet Protocol address).
See Also IP Address.

nsis
: An open-source software tool used to create Windows installers. It is used to create special builds of the Evergreen Staff Client. More information is available at http://nsis.sourceforge.net.

O

OPAC
: The "Online Public Access Catalog"; an online database of a library's holdings; used to find resources in their collections; possibly searchable by keyword, title, author, subject or call number.

OpenSRF
: The "Open Scalable Request Framework" (pronounced 'open surf') is a stateful, decentralized service architecture that allows developers to create applications for Evergreen with a minimum of knowledge of its structure.

P

Perl
: The high-level scripting language in which most of the business logic of Evergreen is written.
See Also CPAN.

PKI
: Public Key Infrastructure (PKI) describes the schemes needed to generate and maintain digital SSL Certificates.
See Also SSL Certificate.

PostgreSQL
: A popular open-source object-relational database management system that underpins Evergreen software.

PuTTY
: A popular open-source telnet/ssh client for the Windows and Unix platforms. As used in Evergreen, a handy utility used to create an SSH Tunnel for connecting Staff Clients to Evergreen servers over insecure networks. More information is available at http://www.chiark.greenend.org.uk/~sgtatham/putty/.
See Also SSH tunnel.

Q

R

Resource Hacker
: An open-source utility used to view, modify, rename, add, delete and extract resources in 32bit Windows executables. It is used to create special builds of the Evergreen Staff Client. More information is available at Resource Hacker

RHEL
: Also known as "Red Hat Enterprises `Linux`". An official `Linux` distribution that is targeted at the commercial market. It is the basis of other popular `Linux` distributions, e.g., `CentOS`. More information is available at http://www.redhat.com.

S

SIP
: SIP (Standard Interchange Protocol) is a communications protocol used within Evergreen for transferring data to and from other third party devices, such as RFID and barcode scanners that handle patron and library material information. Version 2.0 (also known as "SIP2") is the current standard. It was originally developed by the 3M Corporation.

srfsh
: A command language interpreter (shell) that executes commands read from the standard input. It is used to test the Open Service Request Framework (OpenSRF).

SRU
: SRU (Search & Retrieve URL Service) is a search protocol used in web search and retrieval. It expresses queries in Contextual Query Language (CQL) and transmits them as a URL, returning XML data as if it were a web page.
See Also SRW.

SRW
: SRW (Search & Retrieve Web Service), also known as "SRU via HTTP SOAP", is a search protocol used in web search and retrieval. It uses a SOAP interface and expresses both the query and result as XML data streams.
See Also SRU.

SSH
: An encrypted network protocol using public-key cryptography that allows secure communications between systems on an insecure network. Typically used to access shell accounts but also supports tunneling, forwarding TCP ports and X11 connections, and transferring files.

SSH proxy
: As used in Evergreen, a method of allowing one or more Staff Clients to communicate with one or more Evergreen servers over an insecure network by

sending data through a secure SSH tunnel. It also buffers and caches all data travelling to and from Staff Clients to speed up access to resources on Evergreen servers.
See Also SSH, tunneling, SSH tunnel.

SSH tunnel

An encrypted data channel existing over an SSH network connection. Used to securely transfer unencrypted data streams over insecure networks.
See Also SSH, tunneling.

SSL Certificate

As used in Evergreen, it is a method of ensuring that Staff Clients are able to connect to legitimate Evergreen servers.

In general, it is a special electronic document used to guarantee authenticity of a digital message. Also known as a "public key", or "identity" or "digital" certificate. It combines an identity (of a person or an organization) and a unique public key to form a so-called digital signature, and is used to verify that the public key does, in fact, belong with that particular identity.
See Also PKI.

SuperCat

A popular commercial utility used to catalog, search and manage the contents of media such as CDs and DVDs.
See Also PKI.

T

tunneling

As used in Evergreen, it is a method of allowing Staff Clients to securely connect to legitimate Evergreen servers.

In general, it is a method of encapsulating data provided in one network protocol (the "delivery"protocol), within data in a different network protocol (the "tunneling" protocol). Used to provide a secure path and secure communications through an insecure or incompatible network. Can be used to bypass firewalls by communicating via a protocol the firewall normally blocks, but "wrapped" inside a protocol that the firewall does not block.
See Also SSH tunnel.

U

Ubuntu

A popular open-source operating system based on the `Linux` kernel that was originally based on the `Debian GNU/Linux` operating system. More information is available at http://www.ubuntu.com.
See Also Debian, Fedora.

V

Virtual PC

A popular commercial package of virtualization software that emulates the x86 microprocessor architecture. It is installed on a Windows "host" operating system and allows other "guest" (typically including `Linux` and `Windows`) operating systems to be loaded and executed.

See Also Virtualization.

VirtualBox

A popular commercial package of virtualization software that emulates the x86 microprocessor architecture. It can be installed on `Linux`, `Mac OS X`, `Windows` or `Solaris` "host" operating systems and allows other "guest" (typically including `Linux` and `Windows`) operating systems to be loaded and executed.
See Also Virtualization.

Virtualization

A method of executing software in a special environment that is partitioned or separated from the real underlying hardware and software resources. In typical usage, it allows a *host* operating system to encapsulate or emulate a *guest* operating system environment in such a way that the emulated environment is completely unaware of the hosting environment. As used in Evergreen, it enables a copy of the `Linux` operating system running Evergreen software to execute within a `Windows` environment.
See Also VirtualBox, Virtual PC, VMware.

VMware

A popular commercial package of virtualization software that emulates the x86 microprocessor architecture. It can be installed on `Linux`, `Mac OS X`, `Windows` or `Solaris` "host" operating systems and allows other "guest" (typically including `Linux` and `Windows`) operating systems to be loaded and executed.
See Also Virtualization.

Volume Buckets

Virtual “containers” to use in batch processing of multiple volumes. They can be used to perform various cataloging/holdings maintenance tasks in batch.

W

Wine

A popular open-source application that allows `Linux` and `Unix` systems to run `Windows` executables. More information is available at http://www.winehq.org/.

X

XML

The eXtensible Markup Language, a subset of SGML; a set of rules for encoding information in a way that is both human- and machine-readable. It is primarily used to define documents but can also be used to define arbitrary data structures. It was originally defined by the World Wide Web Consortium (W3C).

XMPP

The open-standard communications protocol (based on XML) used for client-server message passing within Evergreen. It supports the concept of a consistent *domain* of message types that flow between software applications, possibly on different operating systems and architectures. More information is available at http://xmpp.org.
See Also Jabber, ejabberd.

xpath

The XML Path Language, a query language based on a tree representation of an XML document. It is used to programmatically select nodes from an XML document and to do minor computation involving strings, numbers and Boolean values. It allows you to identify parts of the XML document tree, to navigate around the tree, and to uniquely select nodes. The currently version is "XPath 2.0". It was originally defined by the World Wide Web Consortium (W3C).

XUL
: The XML User Interface Language, a specialized interface language that allows building cross-platform applications that drive Mozilla-based browsers such as Firefox. More information is available at https://developer.mozilla.org/en/XUL.

xulrunner
: A specialized run-time application environment that provides support for installing, upgrading and uninstalling XUL applications. It operates with Mozilla-based applications such as the Firefox browser. More information is available at https://developer.mozilla.org/en/XULRunner.
See Also XUL.

Y

YAZ
: A programmers' toolkit supporting the development of Z39.50 / SRW / SRU clients and servers.
See Also SRU, SRW, Z39.50.

yaz-client
: A Z39.50/SRU client for connecting to YAZ servers. More information is available at http://www.indexdata.com/yaz/doc/yaz-client.html
See Also SRU.

Z

Z39.50
: An international standard client–server protocol for communication between computer systems, primarily library and information related systems.
See Also SRU.