December 2014 Features Update

Table of Contents

Illustration Index

Global

Although the functional changes this month are not dramatic almost every single file in the site has been changed, particularly to increase the consistency with which different classes perform similar functionality. This makes the implementation easier to follow and avoids potential future programming mistakes.

The database layout that I borrowed from the Legacy Family Tree application on Windows while functional had some features that are problematic on a shared service. I used it because Legacy on Windows could import family tree files from a number of other genealogy products, in particular Family Tree Maker, and the database design was available from the vendor. So I could use Legacy to transform genealogy data from multiple sources into a common form that I could export to the web server. Legacy on Windows was useful for this because unlike some other products it used a relational database supporting Structured Query Language (SQL), in this case Microsoft Access, to record the information. It was therefore easy to mechanically translate the Legacy on Windows database to a different database manager, MySQL, which could be run on a web server.

In designing a relational database it is recommended that the record structure be normalized. Normalization is a systematic way of ensuring that a database structure is suitable for general-purpose querying and free of certain undesirable characteristics, insertion, update, and deletion anomalies, that could lead to a loss of data integrity. In a fully normalized design each unit of information exists in only one place in the database, so changing that information can be done as a simple operation. If the information exists in multiple places then the possibility exists that only some of the locations are changed by an update, creating a situation where multiple different values exist for the same unit of information. This was not an issue for the original implementors of Legacy Family Tree on Windows because their application only supported a single user, whereas a web site based solution needs to consider simultaneous updates by multiple users.

As a specific example the main name and primary birth date are recorded in the basic record for an individual, an instance of `LegacyIndiv`, in each family record to which that individual belongs, instances of `LegacyFamily`, and in the primary name record for the individual, an instance of `LegacyName`. Also the married name of an individual is present both in the family record for the marriage, an instance of `LegacyFamily`, and in a married name record, an instance of `LegacyName`. The name information is therefore moved to exist exclusively in the `LegacyName` record, although it can be logically accessed through either the `LegacyIndiv` or `LegacyFamily` record using existing methods.

Similarly information about facts and events is in general recorded in the table of events, as instances of `LegacyEvent`. However some events, such as the primary birth event and the primary death event, are recorded in the instance of `LegacyIndiv`, while a marriage event is recorded in an instance of `LegacyFamily`, and an LDS sealed to parents event is recorded in an instance of `LegacyChild`. However secondary versions of the birth, death, marriage and other events are recorded in instances of `LegacyEvent`. Some events, such as adoption, do not even have a legitimate method to be recorded in this design since an instance of `LegacyEvent` must be associated with either an instance of `LegacyIndiv` or an instance of `LegacyFamily`, but an adoption should be associated with an instance of `LegacyChild`, since that record identifies both an individual and the family in which that individual is a child, and an adoption must be associated both with the

individual and the adoptive parents. Therefore all fact and event information is now recorded in instances of `LegacyEvent`, which may also be associated with an instance of LegacyChild, providing support of adoption and LDS sealed to parents events.

There are 4 textual notes in `LegacyIndiv` and one in LegacyFamily that can have citations. In the future those will be represented by separate instances of a new class `LegacyNote`. This will also permit modeling more completely the implementation of textual notes in the industry standard Genealogical Data Communication file transfer format (GEDCOM 5.5). GEDCOM 5.5 permits more than one note of a particular class associated with an entity in the family tree which, among other advantages, permits each note to have separate citations. Most user interfaces provided by genealogy applications do not actually support multiple notes of a particular class because that would complicate the user interface and make the application harder to use.

This restructuring significantly simplifies the implementation of a source citation, represented by an instance of `LegacyCitation`. In the original implementation each instance of `LegacyCitation` contains a field `` `Type` `` which in addition to identifying the type of record containing the fact or event, also identifies which fact or event within the associated record the citation applies to. For example there are 8 different events and 5 different facts within an `LegacyIndiv` instance alone that can have citations, including the birth event and the primary name of the individual. Moving all of the events in `LegacyIndiv`, `LegacyFamily`, and `LegacyChild` into instances of `LegacyEvent`, recognizing that the primary name is already unnecessarily duplicated from the primary instance of `LegacyName` for the individual, which can already have citations, and adding the proposed `LegacyNote` class, leaves no remaining facts in the `LegacyIndiv` record, and only two flags in `LegacyFamily`, and three status fields in `LegacyChild` that require being distinguished.

The script `confirmUserXml.php`, which is invoked when a new user clicks on the link in the confirmation e-mail, is updated to output diagnostic information when requested by the debug option to the variable `$warn`, and to include that diagnostic information in the XML response to Javascript.

The script `deleteUser.php`, which is used by the administrator to manually delete a registered user, is updated to output diagnostic information when requested by the debug option to the variable `$warn`, and to include that diagnostic information in the XML response to Javascript.

Family Tree Enhancements

The handling of events in the page `editIndiv.php` has been extensively rewritten. The key driver for this was to make the function of the "Reorder Events by Date" button much faster. This button now updates the display, and the events are not reordered in the database until you save the individual. Since the primary events for an individual are being moved the handling of these events is now the same whether they are actually stored inside the instance of `LegacyIndiv` or in separate instances of `LegacyEvent`. This means that the instance of `LegacyEvent` now distinguishes between the preferred version of each event, and any other instances by means of a "Preferred" flag.

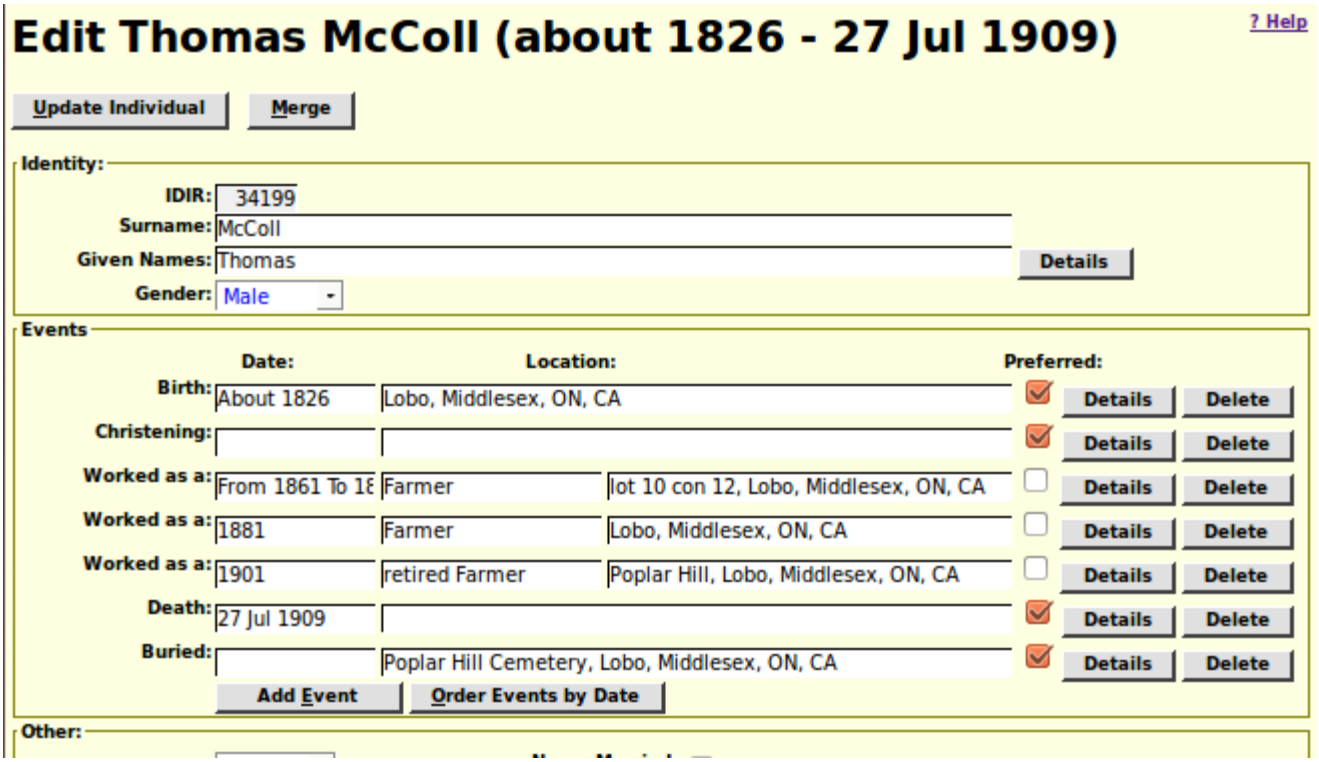

Illustration 1: Editing with Preferred Events

You cannot click on the "Preferred" check-box to turn the flag off, because that would leave no preferred instance of the class of events. However if there are multiple instances of a particular event type, for example multiple birth events, then you can click on the unchecked check-box of another instance and make it the preferred instance.

Classes `LegacyAddress`, `LegacyChild`, `LegacyCitation`, `LegacyDontMergeEntry`, `LegacyEvent`, `LegacyFamily`, `LegacyHeader`, `LegacyIndiv`, `LegacyLocation`, `LegacyName`, `LegacyPicture`, `LegacySource`, `LegacySurname`, and `LegacyTemple` are changed to redirect diagnostic information to `$warn`.

The functionality of ensuring that the names and birth dates of the spouses in a family match the names and birth dates in the associated `LegacyIndiv`, `LegacyEvent` and `LegacyName` records is moved to `LegacyFamily::__construct`. Through database normalization the only authoritative copy of the primary name of the individual is not in the instance of `LegacyName` identified by

Order 0. The only authoritative version of the primary birth date of the individual is in the preferred instance of LegacyEvent with event type 'birth'.

The method LegacyChild::getParSealEvent is added so that the information about the LDS sealed to parents event in the LegacyChild record can be managed as an instance of LegacyEvent. The existing fields 'parseald', 'parsealsd', 'idtrparseal', 'parsealnote', and 'ldsp' are deprecated and should not be accessed using LegacyChild::getField, but rather as the 'eventd', 'eventsd', 'idlrevent', 'desc', and 'ldstempleready' fields of the instance of LegacyEvent returned by LegacyChild:: getParSealEvent.

The constructor for the class LegacyEvent is enhanced to support the LDS sealed to parents event. Instances of LegacyEvent may now be associated with instances of LegacyChild to support the LDS sealed to parents event, and the adoption event.

The method LegacyEvent::getDate now accepts a privacy year limit parameter.

The class LegacyFamily adds a method LegacyFamily::getSealEvent which permits accessing the LDS sealed to spouse event as an instance of LegacyEvent. The existing fields 'seald', 'sealsd', 'idtrseal', 'sealnote', and 'ldss' are deprecated and should not be accessed using LegacyFamily::getField, but rather as the 'eventd', 'eventsd', 'idlrevent', 'desc', and 'ldstempleready' fields of the instance of LegacyEvent returned by LegacyFamily:: getSealEvent.

For improved performance references to the instances of LegacyIndiv for the two spouses are kept in the LegacyFamily instance.

Additional diagnostic output is generated by LegacyFamily::__construct.

The static method LegacyFamily::getFamilies now supports the limit and offset parameters for consistency with other classes. The associative array returned by this method is now indexed by the key IDMR. The method now takes a third parameter which, if true, results in XML output being generated if this method is called from a script that produces XML.

The LegacyIndiv methods to get preferred instances of LegacyEvent now uses the associative array of parameters method to invoke the constructor for LegacyEvent to make the code easier to maintain and understand.

The class LegacySource has added a method LegacySource::getCitations which provides an associative list of instances of LegacyCitation that reference this instance of LegacySource.

If the browser is pointed at the folder /FamilyTree/testscripts on the site the browser is redirected to display the menu of test scripts if the current user is an administrator, or else an error message, instead of the contents of the folder.

The script addCitXml.php which is invoked by Javascript on the browser to add a citation to a fact or event is enhanced to ignore zero length parameter values. Its invocation of LegacyName:: getNames is changed because the function signature has changed to permit feeding back the number of matching names.

The script mergeUpdIndivid.php is changed because the method LegacyIndiv::getFamilies now returns an associative array indexed by IDMR.

The script orderMarriagesByDate.php is renamed to orderMarriagesByDateXml.php. It is enhanced to validate that the IDIR parameter passed to it is numeric and greater than 0.

The script mergeIndivid.php is enhanced to pass the debug flag to the script mergeUpdIndivid.php.

The script editMarriages.php is changed because the method LegacyIndiv::getFamilies now returns an associative array indexed by IDMR.

The script editParents.php is changed because the method LegacyIndiv::getParents now returns an associative array indexed by IDMR.

The script editEvents.php is changed to add a row number parameter to the eventFeedback method of the invoking form.

The script editIndivid.php is significantly changed to support the new model of events. All events and facts now have the same layout, with the exception of the names for the date, location, and description field which are different for some events in order to display field specific popup documentation, and induce the browser auto-complete functionality to treat them distinctly. Each row now has the following additional fields, most of which are hidden from the user if the debug option is not specified:

- checkbox for preferred status
- the IDER value (which is zero for events that are still in LegacyIndiv)
- the IDET value
- the citation type (usually 30)
- the order in which the events are displayed
- the sorting form of the event date (so browser resident code can reorder the events)
- the event changed indicator

The script deleteSourceXml.php is enhanced to ignore a request to delete any master source that still has citations referencing it. The script that invokes this does not provide a delete button, but I always believe in the value of both a belt and suspenders.

The script descendantReport.php is changed because the method LegacyIndiv::getFamilies now returns an associative array indexed by IDMR.

The script ancestorReport.php is changed because the method LegacyIndiv::getParents now returns an associative array indexed by IDMR.

The script legacyIndivid.php, which displays the main page for an individual, is changed to always use LegacyIndiv::getBirthEvent and LegacyIndiv::getDeathEvent to get the birth and date information to display, particularly in the title of the page. The script is also changed because the methods LegacyIndiv::getFamilies and LegacyIndiv::getParents now

return an associative array indexed by IDMR. The script also now handles an exception thrown if the wife's IDIR was invalid.

The script `getIndivSvg.php` which displays a graphical family tree is substantially rewritten. The use of deprecated methods of `LegacyFamily` is brought up to date. The XML and PHP code is separated, which makes the implementation easier to maintain. Unique id attribute values are now supplied for all SVG elements. Error messages, for example to report bad parameters to the script, are now displayed in bold red text. A title is not displayed for the page including the name and dates of the key individual.

The script `getFamilyOfXml.php` is changed because the method `LegacyIndiv::getFamilies` now returns an associative array indexed by IDMR. It is also changed to use `LegacyIndiv::getBirthEvent` to get birth information.

The script `deleteNameXml.php`, used to delete an instance of `LegacyName` from the database, is enhanced to ignore a request to delete a primary name record if the associated instance of `LegacyIndiv` is still present, or a married name record if the associated instance of `LegacyFamily` is still present in the database.

The script `addFamilyOfXml.php` is changed to use `LegacyIndiv::getBirthEvent` to get birth information.

The script `addDontMergeXml.php` is changed to include the actual SQL command used to update the database, and the contents of the inserted record in the response.

The script `detChildXml.php`, which permits browser code to delete an instance of `LegacyChild` from the database, is changed to perform additional parameter validation, to include the contents of the child record being deleted, include the actual SQL command used to update the database, and include diagnostic messages if present as a result of the debug option.

The script `orderEventsByDate.php` is obsolete and no longer performs any function.

The script `GedcomUpdate.php` is not yet functional, but it has been updated to use current interfaces to the various classes.

The script `updateEvent.php`, which is used by browser code to update an event record in the database, is changed to use `LegacyIndiv::getBirthEvent` to get birth information.

The script `getIndivNamesXml.php`, which is used by browser code to get information to populate selection lists of individuals, is changed to use `LegacyIndiv::getBirthEvent` to get birth information. It also now handles an exception thrown by `LegacyFamily` for a bad spouse IDIR value.

The browser script `locationCommon.js`, which provides automatic completion assistance for fields containing values that map to instances of `LegacyLocation`, is enhanced to support any field whose name contains the text 'Location'.

Census Database Enhancements

In the past when you clicked on the button to display the original census image, a new window was opened to display the image. This then required that the user rearrange and resize the two windows in order to take advantage of the full screen space available. Through years of experimentation I have found that the most efficient way to transcribe censuses is a column at a time, with the screen split with the input form using one half of the screen and the original image using the other half of the screen, side by side. The implementation of the display image button is therefore changed to facilitate this. Just make the census window full screen, and clicking on the display image button splits the screen for you into left and right halves.

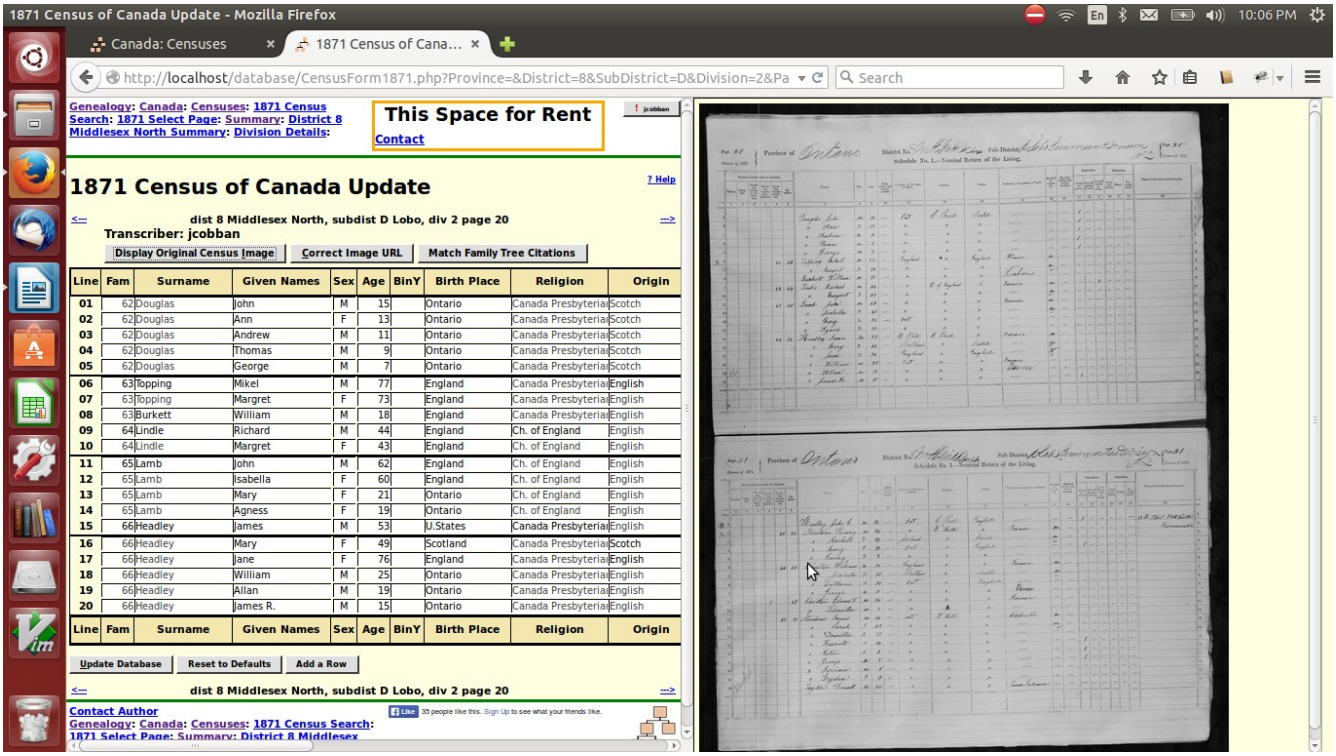

Illustration 2: Split Screen Census Transcription Editing

Classes `CensusLine`, `District`, `Pages`, and `SubDistrict` are changed to redirect diagnostic information to `$warn`.

The implementation of the class `District` is changed to improve the clarity of the implementation of `District::getField`, improve the clarity of the implementation of `District::setField`, and improve the efficiency of `District::getSubDistricts` by saving the previously obtained value, so subsequent calls do not require accessing the database. Methods `District::getSdCount` and `District::getFCount` are removed. The information these methods returned is still available through `District::getField` with field names '`d_sdcount`' and '`d_fcount`' and these two methods were unused.

The implementation of the class `SubDistrict`, representing a single enumeration division, is enhanced. If an instance of `District` is passed to the constructor then the census identifier from that

instance is used. A new method `SubDistrict::getPages` is added to obtain an array of instances of `Page` for the sub-district.

The implementation of the class `Page`, representing a single page from a census, is enhanced. The constructor now accepts an associative array of parameters, while remaining backwards compatible. Invoking the constructor using an associative array makes the invocation more self-explanatory. A static method `Page::getPages` is added to encapsulate obtaining an array of instances of `Page`.

If the browser is pointed at the folder /`database` on the site the browser is redirected to display the menu of censuses of Canada instead of the contents of the folder.

If the browser is pointed at the folder /`database/testscripts` on the site the browser is redirected to display the menu of test scripts if the current user is an administrator, or else an error message, instead of the contents of the folder.

Vital Statistics Enhancements

Classes Birth, County, Death, Marriage, MarriageParticipant, and Township are changed to redirect diagnostic information to $warn.

The table recording information about Counties is changed to remove the column Prov which is redundant since the same information is contained in the column Domain. To ensure no loss of functionality in code which uses the class County the method County::getField returns the correct value derived from the column Domain. The static method County::getCounties is changed to conform to the behavior of similar methods in other classes by accepting the parameters limit and offset on input and return the count of matching rows in the 'count' field of the parameter array.

All of the pages on the site that are specific to Canada, with the exception of the pages that apply only to Ontario and for censuses, are moved to a new folder /Canada. This includes the pages genCanMilitary.html, genCanCom.html, genBritCol.html, and genManitoba.html. For provinces that do not have a specific page of links a generic page, created by the new script /Canada/genProvince.php, is implemented to provide the ability to transcribe and search vital statistics records for those provinces.

The pages on the site that are specific to the United States of America are now all moved to the folder /USA.

The scripts BirthRegDetail.php, BirthRegStats.php, BirthRegQuery.php, BirthRegResponse.php, and BirthRegUpdate.php are enhanced to support displaying and updating the transcriptions of birth registrations for all provinces, and are consequently moved from the /Ontario folder to the /Canada folder. The former page BirthRegQuery.html, which only supported Ontario, is replaced by BirthRegQuery.php?domain=CAON which generates a query page specific to any defined domain.

MarriageRegDetailLib.php is changed to use LegacyIndiv::getBirthEvent and getDeathEvent because those events are no longer always present within the instance of LegacyIndiv.

If the browser is pointed at the folder /Canada on the site the browser is redirected to display the menu of services for Canada instead of the contents of the folder. If the browser is pointed at the folder /USA on the site the browser is redirected to display the menu of services for the United States of America instead of the contents of the folder. If the browser is pointed at the folder /Ontario on the site the browser is redirected to display the menu of services for Ontario instead of the contents of the folder.

If the browser is pointed at the folder /Ontario/testscripts on the site the browser is redirected to display the menu of test scripts if the current user is an administrator, or else an error message, instead of the contents of the folder.

The response from the script deleteDeathRegXml.php now includes the number of records deleted. This is 1 if the parameters identified an existing record, or 0 if there was no existing record.

When a user submits updates to the transcription of a birth or death registration and there are no error

messages or debugging output to display to the user, the form to update the next sequential birth or death registration is displayed instead of a dialog asking the user what to do next. This facilitates transcribing a series of birth or death registrations.

Bug Fixes

- Method `Record::dump` did not correctly encode the values of text fields for the case where it was called to generate debugging information in an XML document.
- The method `LegacyIndiv::getEvents` failed if called for a new instance before it was associated with a record in the database.
- The method `LegacyEvent::delete` did not pass the second parameter to the method `LegacyCitation::deleteCitations`. This meant that information about the deletion of citations associated with an event did not appear in the XML output of the script `FamilyTree/deleteEventXml.php`.
- The static method `LegacyIndiv::getWhere` generated a syntactically invalid WHERE clause if invoked with both the IDIR and limit parameters.
- The static method `LegacyIndiv::getIndivs` returned a count of 0 if it was invoked with an IDIR parameter and limit set to 0. It now returns `$parms['count']=1` if the supplied IDIR exists in the database and 0 if it does not.
- When adding early childhood events, such as christening, they were added after all midlife events, rather than between birth and the first midlife event by the methods `LegacyIndiv::getBaptismEvent`, `LegacyIndiv::getConfirmationEvent`, `LegacyIndiv::getInitiatoryEvent`, and `LegacyIndiv::getChristeningEvent`.
- In the output of `LegacyLocation::dump` the label was incomplete.
- The constructor `LegacyLocation::__construct` called itself if invoked with an associative array of parameters.
- The exception thrown by the static method `LegacyPicture::getPictures` identified the wrong method name.
- There was an invalid operator in the SQL generated by the script `orderChildrenXml.php`. It is fixed, but this script is no longer used because events are now reordered only by `updateIndividXml.php`.
- The global `$debug` was not declared in function `getDateAndLocation` in script `editEvent.php`.
- There was a missing parameter to `LegacyTemple::getTemples` in script `editEvent.php`.
-