

Tips For Displaying Related-Table Data

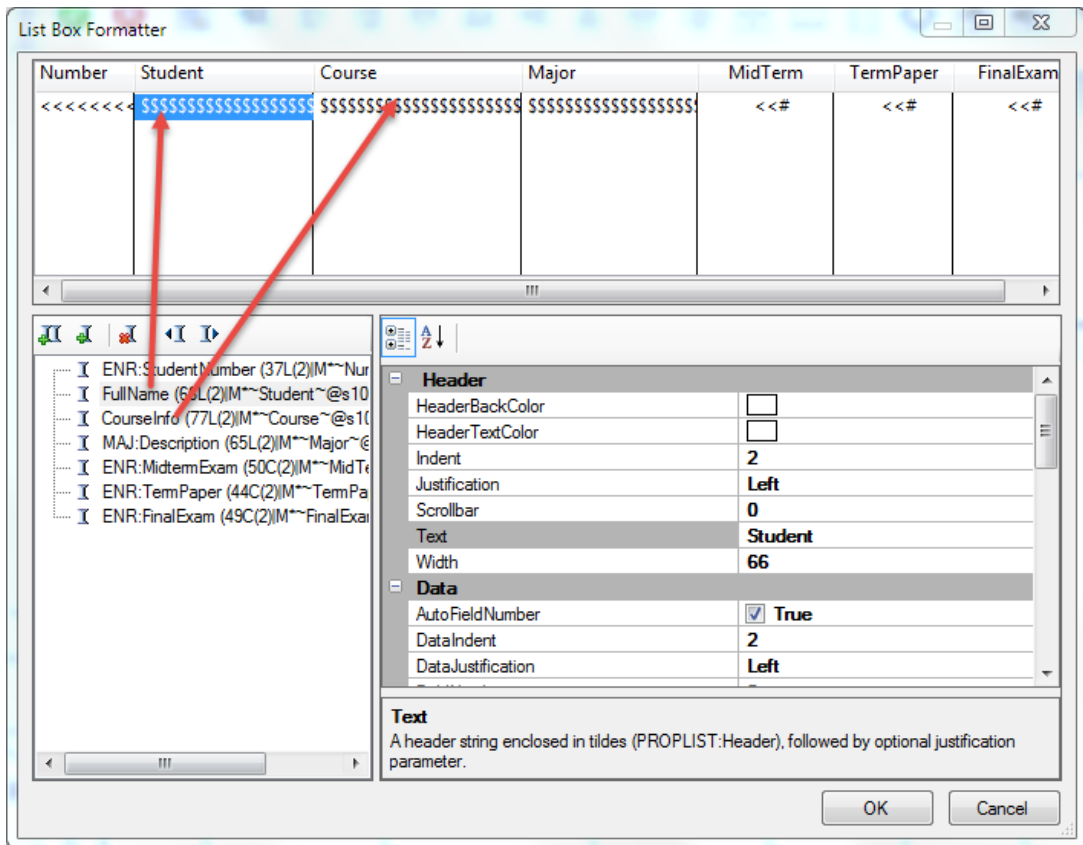
On Clarion, ABC Browsers And Forms

Related-Table-Data On ABC BROWSES

First, on the topic of related-table-data displaying on ABC Browsers and Forms, we'll explain how it's possible to display local variables containing concatenated field data from related tables into an ABC browse and have the browse still sort, search and filter the columns containing them as if they were data fields from the browse VIEW.

The information on this browse originates in FIVE related SQL tables.... To save space, two columns, STUDENT and COURSE display concatenated values, yet they are header-click sortable and they allow searching and filtering. HOW DOES THAT WORK?

Number	+Student	Course	Major	MidTerm	TermPaper	F
768903748	Babbitt, Jim	Sociology 100 [MWF 11]	Computer Science	0	0	
831877376	Babin, Brian	Calculus I [M-W 900]	Computer Science	0	0	
251058624	Bachman, Robert F	Calculus I [T-TH 1100]	Computer Science	99	96	
967263524	Bagby, John H	English Composition2 [T-Th 1100]	Law	96	90	
141027461	Bailey, Alvin L	Microcomputers [M-W 900]	Computer Science	100	97	
286959466	Bair, Curtis	English Composition2 [T-Th 1100]	Law	88	93	



Below, using a combination of pictures and words we'll outline how that's done using a standard ABC browse enhanced with CHT HandyMarkerBrowse -- though what we're showing here works just the same using CHT ExplorerBrowse or CHT LocatorOverrideControl.

IMPORTANT: Clarion VIEW Structures - What They Do

A Clarion VIEW structure is generated into your procedure code when you drop an ABC browse template on any ABC window and populate fields into the browse, or add "Hot Fields" for the purpose of concatenating to locals.

Below is an image of just such a view generated for the **BrowseEnrollments()** procedure in our SQLite demo HND SCHOOL.APP, a demo we constructed based on the modified SCHOOL.DCT and SCHOOL.APP demo provided by SoftVelocity. Its also important to note that Clarion views are ISAM/SQL agnostic. They're meant to work the same way, at least from the developer's perspective, on an ISAM file as on an SQL table.

Quite a number of years ago, when CHT first introduced browse column sorting on ISAM data columns without file keys, Clarion developers wondered how we were doing this! Well that little miracle was based on the design incorporated into Clarion's VIEW structure from the outset. Views may have a PROP:ORDER property assigned to them as we explained above. That means the view SORTs the incoming data-set according to the order property being applied whether the source file or table has a key on the designated sort-order column(s) or not.

At the time, well before we all began to migrate our applications to use SQL, the concept of keyless, indexless order columns had long been a given! But to Clarion guys, familiar only with ISAM files, it was a revelation. As it turns out, Clarion's file drivers were sorting this ISAM data-set internally, and on the fly, whenever a keyless column was assigned to PROP:ORDER an incoming view-based data set. It wasn't necessarily always blazingly fast, but on smallish, local data tables it changed the Clarion browse design paradigm for us at CHT dramatically.

Clarion VIEWS are actually based on the concept of SQL VIEWS which similarly treat multiple, related tables as a single, navigable flat file. Bruce Barrington, the father of several early Clarion versions, wrote about this extensively when the concept was first incorporated into Clarion file drivers and made to apply to ISAM as well as SQL in a transparent-to-the-developer kind of way.

CHT Development staff realized - and leveraged with several ABC browse extensions - the magic of Clarion VIEWS long before other 3rd-party toolmakers did, even well before Clarion as SoftVelocity introduced the concept of multi-column browse sorting into their base browse template.

```

ue for Enrollment
VIEW(Enrollment)
  PROJECT (ENR:StudentNumber)
  PROJECT (ENR:MidtermExam)
  PROJECT (ENR:TermPaper)
  PROJECT (ENR:FinalExam)
  PROJECT (ENR:ID)
  PROJECT (ENR:ClassNumber)
  JOIN (STU:KeyStudentNumber, ENR:StudentNumber)
    PROJECT (STU:LastName)
    PROJECT (STU:FirstName)
    PROJECT (STU:Major)
    PROJECT (STU:Number)
  JOIN (MAJ:KeyNumber, STU:Major)
    PROJECT (MAJ:Description)
    PROJECT (MAJ:Number)
  END
END
JOIN (CLA:KeyClassNumber, ENR:ClassNumber)
  PROJECT (CLA:ScheduledTime)
  PROJECT (CLA:ClassNumber)
  PROJECT (CLA:CourseNumber)
  JOIN (COU:KeyNumber, CLA:CourseNumber)
    PROJECT (COU:Description)
    PROJECT (COU:Number)
  END
END
END
END

```

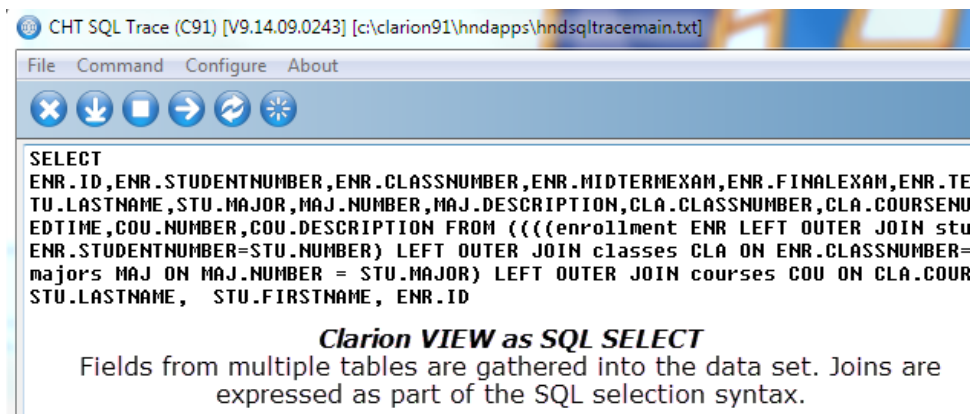
This is the BrowseEnrollments procedure's underlying VIEW. It expresses all of the relationships shown in the browse file tree. Even though it is composed of multiple, related files, it acts like a single, flat file. From a Clarion app this VIEW can be navigated with NEXT() and PREVIOUS() just as if it were a single file. A VIEW can incorporate FILTER and ORDER properties. Fields shown in the VIEW are fetched from its various tables into the local file buffers as if you had fetched the individual files separately.

As explained in the word-balloon on the image, a *Clarion VIEW* is a structure that describes a set of table interrelationships. This structure can be OPENed and CLOSEd like a file and traversed UP and DOWN respectively with PREVIOUS() and NEXT().

Clarion VIEW structures also have a PROP:FILTER property that accepts an expression which causes it to logically select certain records from the tables while rejecting others. There is also a PROP:ORDER property which accepts a direction setting as well as a column name or column names, which determine the order in which the view is processed moving either forward or backward through the source table(s).

To your code, as a VIEW is traversed, it looks like a flat file. The fields come in when you NEXT() through the view as if they were coming from one table instead of from multiple tables. The built-in wonders of Clarion file driver code handles all of the details of keeping the various files or tables underlying the view in synch, assuming, of course, that they've been properly joined as all good, normalized data should be.

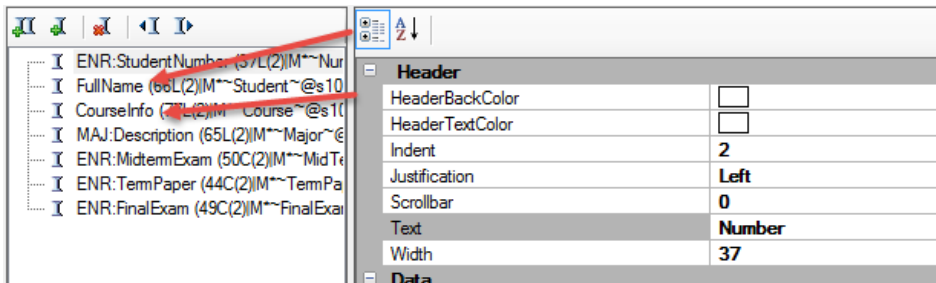
In the next screen shot, is an SQL SELECT (issued from Clarion's SQLite Driver) for the same Clarion VIEW structure pictured above. Note that the image is somewhat clipped and therefore incomplete. But the concept should be clear enough.



Clarion VIEW as SQL SELECT
Fields from multiple tables are gathered into the data set. Joins are expressed as part of the SQL selection syntax.

Concatenating VIEW Data For Insertion To The Browse

Near the beginning of this article is a screen-snap of two local variables Fullname and CourseInfo which have been inserted into our BrowseEnrollments() procedure. Here is part of that image again to refresh your memory.



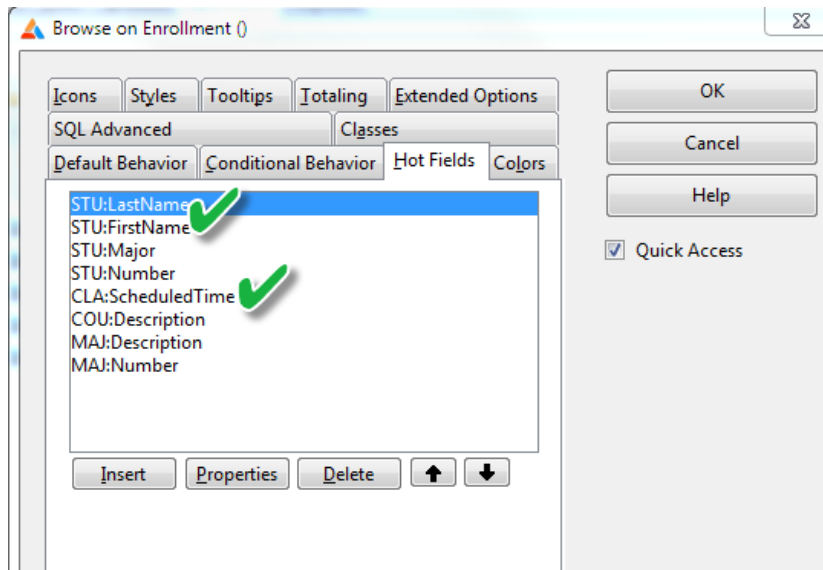
These two locals contain four VIEW fields from three different underlying tables. The purpose for doing this via formatted concatenation, *beside the fact that we can*, is to provide a maximum of information on the browse using a reasonable amount of browse list box real estate.

By the way "because we can" is never a good criterion to apply when implementing any "nifty" new technical tool or skill. If pigs could be made to fly, probably the only outcome would be sky-high bacon prices.

The FullName variable, applied to the "Student" column consists of the field STU:LastName, a comma, a space plus the field STU:FirstName from the "Students" data table.

The CourseInfo variable, applied to the "Course" column consists of the field COU:Description, an opening square bracket, a space, plus the field CLA:ScheduledTime from the "Classes" data table, then closing square bracket.

We've made sure via "ABC HotFields" that these three fields (amongst others) were included in the VIEW structure - *check and see for yourself, above, if you feel like it* - so they are available for use in those parts of the procedure where ABC is populating browse data to the list box.



The concatenation code is embedded into a procedure embed point located in an ABC Browse Class function called SetQueueRecord(). This is the same place that ABC incorporates listbox customizations such as color and font into the list box structure. Below is the actual code:

```

BRW1.SetQueueRecord PROCEDURE

! Start of "Browser Method Data Section"
! [Priority 5000]

! End of "Browser Method Data Section"
CODE
! Start of "Browser Method Code Section"
! [Priority 100]

!Region Start Template Embed: (Browser Method Code Section) CHT
CLEAR(SELF.Q.Mark)
!EndRegion End Template Embed: (Browser Method Code Section) C

! [Priority 1400]

CourseInfo = CLIP(COU:Description) & '<32>[' & CLIP(CLA:ScheduledTime) & ']'
FullName = CLIP(STU:LastName) & ', ' & STU:FirstName
! [Priority 3800]

! Parent Call
PARENT.SetQueueRecord
! [Priority 5500]

```

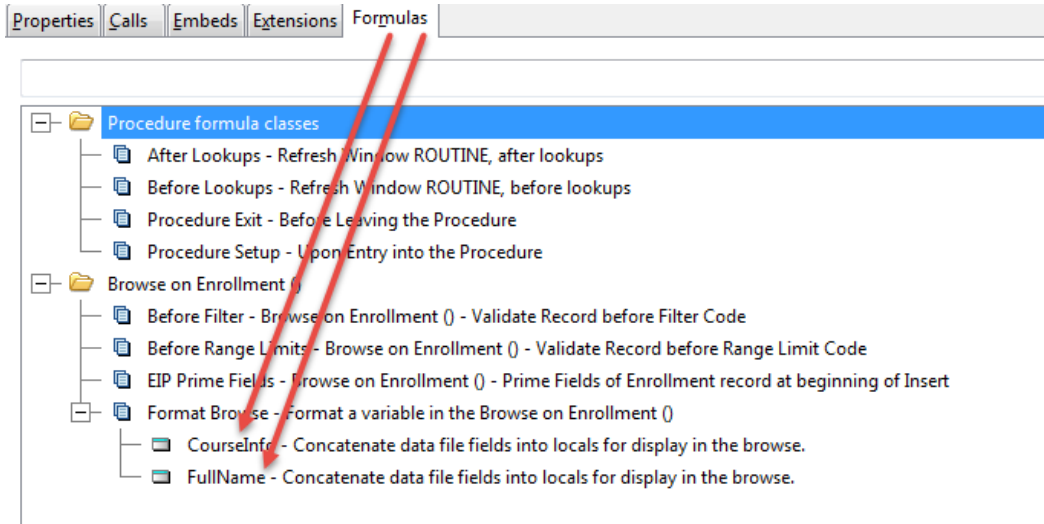
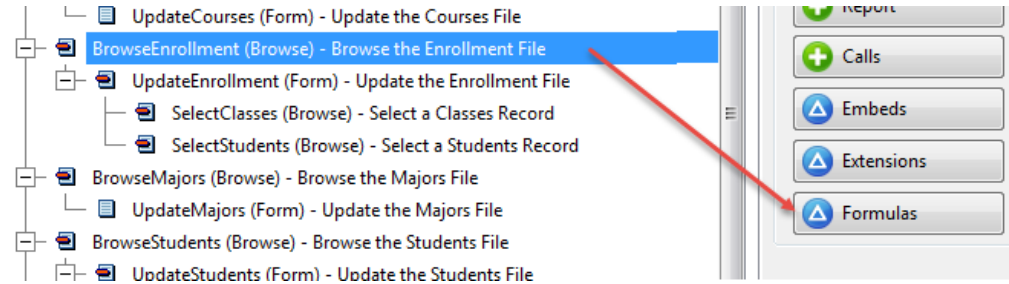
These two concatenated strings are inserted into the browse in the "Student" and "Course" columns. This specific "embed" example is accomplished by using the IDE's Formulas Editor. A hand-embed is probably faster and just as easy now that Clarion has code-assist. The embed should be placed at [Priority 1400].

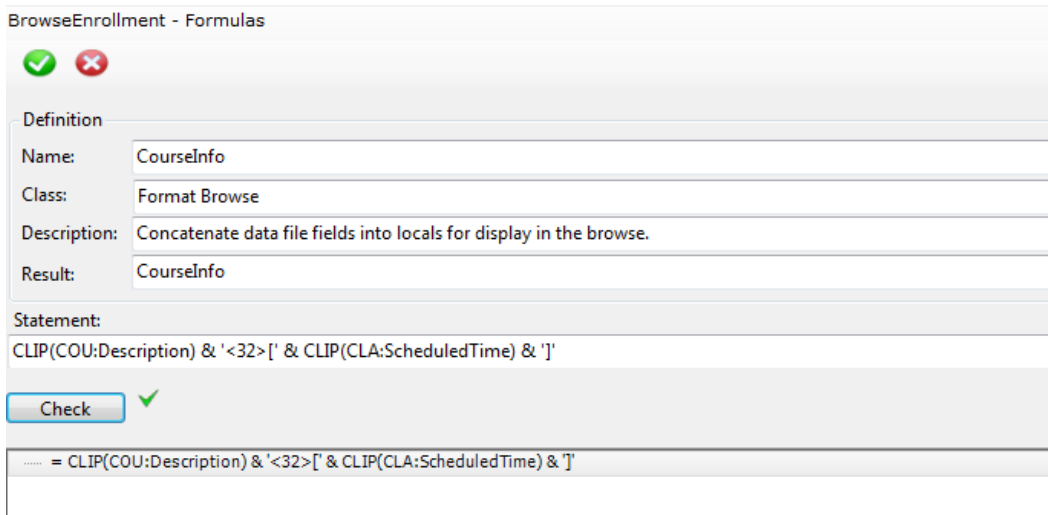
```

CourseInfo = CLIP(COU:Description) & '<32>[' & CLIP(CLA:ScheduledTime) & ']'
FullName = CLIP(STU:LastName) & ', ' & STU:FirstName
! [Priority 3800]

```

As the text-balloon indicates, this code is not "hand-embedded", it was inserted into this position [Priority 1400] in the browse class SetQueueRecord() method using the IDE's Formulas Editor. The next sequence of images illustrates how that works.





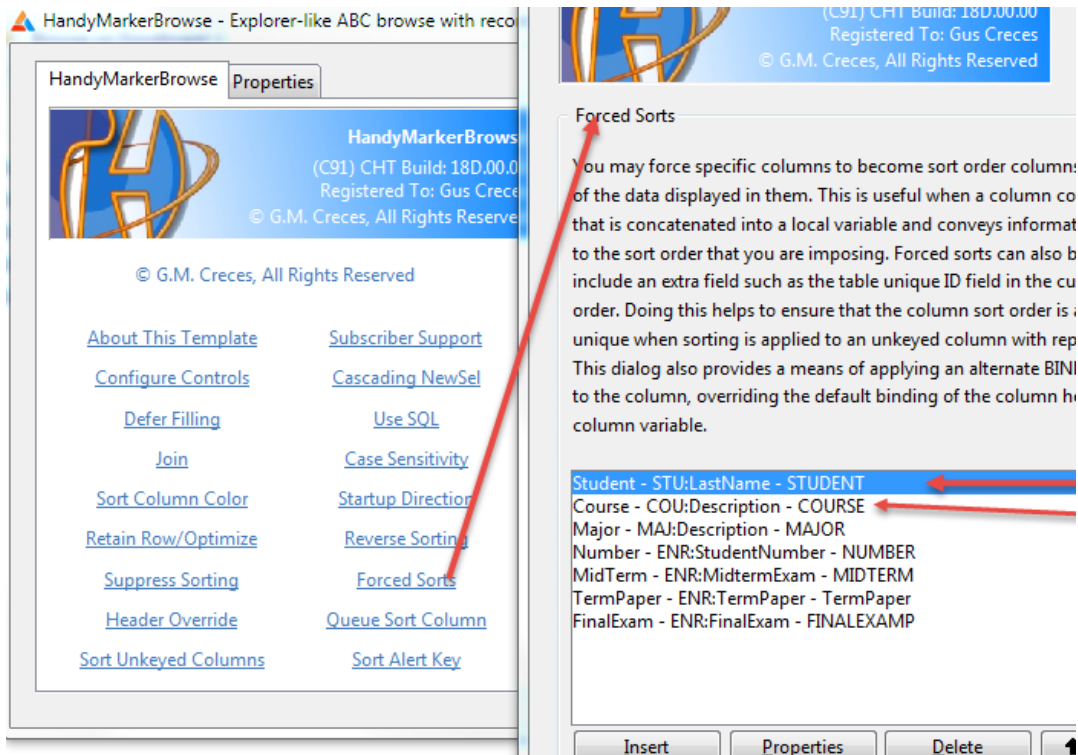
FINAL STEP: Sorting And Searching Columns With Local Variables

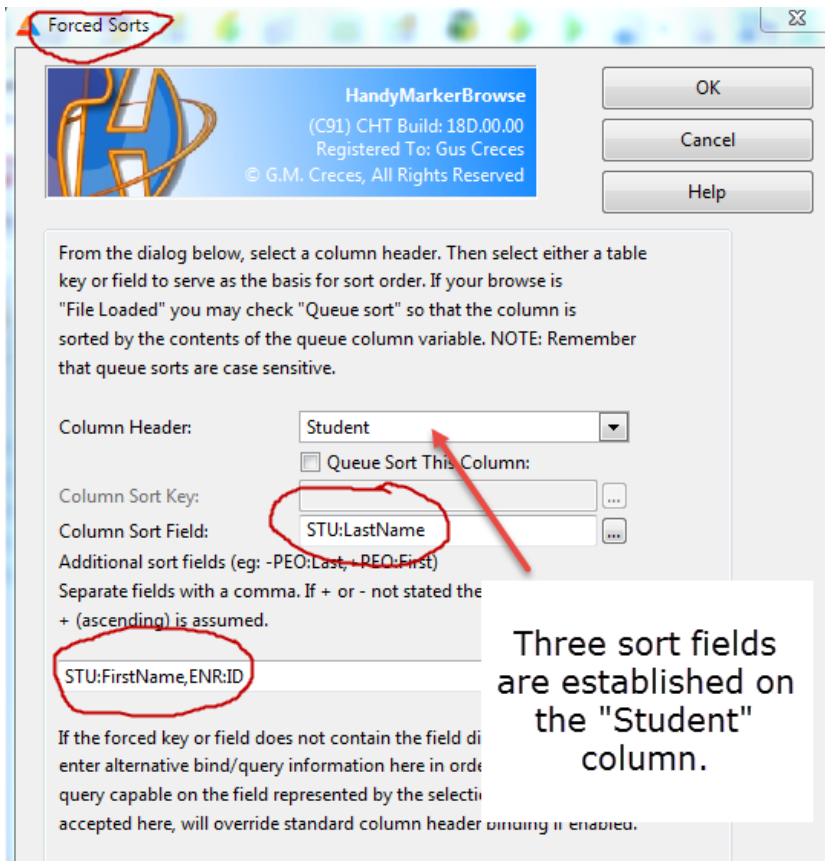
This part of the implementation relieves the developer of having to code anything at all, when they are using a CHT browse extension as we are, in this case, a CHT *HandyMarkerBrowse*.

All of our ABC browse extension templates provide a dialog called "Forced Sorts" where it is possible to cause a local variable to act as if it was a table variable. This happens again through the wonders of the Clarion VIEW structure with its PROP:FILTER and PROP:ORDER properties.

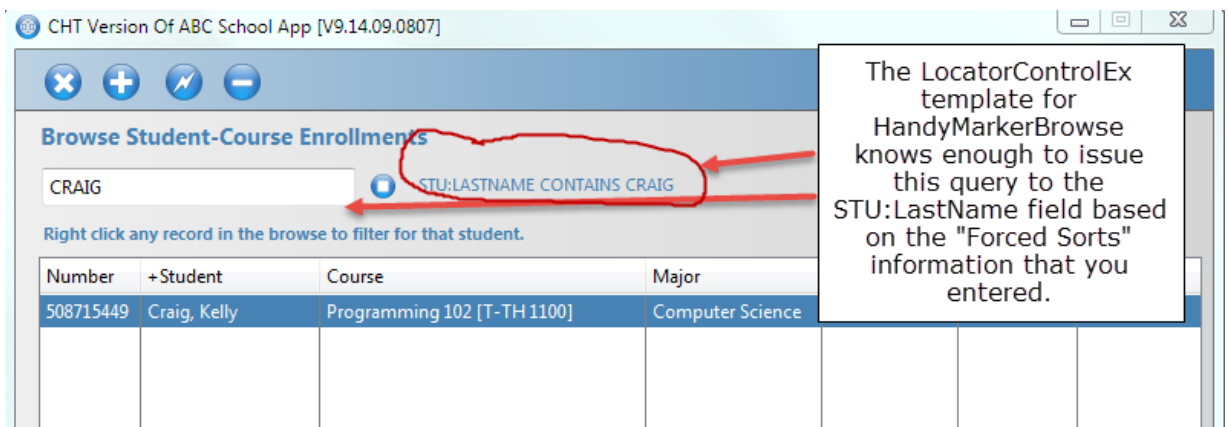
The CHT "Forced Sorts" dialog looks as in the following screen-snap. Here you indicate to our template, *which then writes the code for you*, which back end data field should be surrogated whenever the user *ORDERS* or *FILTERS* the *VIEW* by this local-variable column.

Easy as pie, right? You tell the template which table variable(s) to sort the view by and CHT code issues the correct PROP:ORDER to the view on your behalf. That's exactly what ABC does when you populate a table variable to the browse. CHT code just intervenes on your behalf whenever you insert a local that has been "Force Sort" adapted to surrogate for a designated table field.





This same "Forced Sorts" dialog tells the CHT search and query mechanism that the same column should be searched and queried using this back-end table variable whenever queries and locates are issued from one of our many plug-in templates to one of these columns containing *local* display variables.



The SQL output generated by Clarion's VIEW in combination with the Clarion SQLite file driver, filters and orders the browse as if the "Student" column contained STU:LastName instead of local variable "StudentInfo".

```

SELECT
ENR.ID,ENR.STUDENTNUMBER,ENR.CLASSNUMBER,ENR.MIDTERMEXAM,ENR.FINALEXAM,ENR.TERMPAPER,STU.NUMBER,
STU.LASTNAME,STU.MAJOR,MAJ.NUMBER,MAJ.DESCRPTION,CLA.CLASSNUMBER,CLA.COURSENUMBER,CLA.TEACHERI
EDTIME,COU.NUMBER,COU.DESCRPTION FROM (((enrollment ENR LEFT OUTER JOIN students STU ON
ENR.STUDENTNUMBER=STU.NUMBER) LEFT OUTER JOIN classes CLA ON ENR.CLASSNUMBER=CLA.CLASSNUMBER)
majors MAJ ON MAJ.NUMBER = STU.MAJOR) LEFT OUTER JOIN courses COU ON CLA.COURSENUMBER=COU.NUM
WHERE ( UPPER(STU.LASTNAME) LIKE '%CRAIG%' ) ORDER BY STU.LASTNAME, STU.FIRSTNAME, ENR.ID
----- NEXT READ ----- 18:20:52 ----- NEXT READ -----

```

The CHT browse extension's, PROP:ORDER property was further improved by subsorting on back-end field STU:FirstName to ensure correct sortation in the event of multiple, identical last names. And finally, a subsort was added to include ENR:ID, the Enrollment table unique ID, to ensure column uniqueness for refetch, during a reset.

This last little detail fixes an *ABC browse reset, duplicate record display bug with SQL implementations* by ensuring that this column's refetch criterion is unique to the back-end table being queried. This stops the ABC Browse Class from appearing to duplicate this record in the browse when the browse resets by giving SQL something with which to positively identify the browse record by including a unique ID field in the sort/fetch properties applied to the Clarion VIEW.

That's all for this topic.

To review, in brief, it is *possible* with all CHT browse extension templates to insert local variables into the list box queue and have the ABC browse operate when querying and ordering, as if the column containing that variable was a back-end table field.

Related-Table-Data On ABC FORMS

If you've been following our forum posts about SQLite and HNDSCHOOLAPP, the idea of publishing this "tip" at this time, came up as we were working on a CHT demo called, HNDSCHOOL.APP. It's a way to display related-file information on a standard ABC update form with a minimum of code. This technique pertains to all ABC forms regardless of back-end data file types, ISAM or SQL. It's something we've practiced for years.

We've chosen to incorporate this technique, once again, in the HNDSCHOOL.APP SQLite application, as the data tables are fairly highly "[normalized](#)". We should let you know beforehand, that we didn't design these data layouts, so the field and key naming conventions are a bit counter intuitive, but workable, nevertheless.

The Enrollment table on which we're building an ABC edit form, in particular contains two joins to other tables and not much "displayable" information, only the IDs of related tables. When editing a form based on what is essentially a "join table" it's always helpful to the user to display something from the related tables to help orient them. You'll see what we mean in a minute.

Here, first, are the Enrollment and Student table definitions:

ENR:StudentNumber is a join field from the Students table primary ID STU:Number
ENR:ClassNumber is a join field from the Classes table primary ID CLA:ClassNumber

The Classes table joins to the "Courses" table via `CLA:CourseNumber = COU:Number` and a class "Scheduled Time" needs to be displayed from `CLA:ScheduledTime` while the Courses table contains the "Course Description" in `COU:Description`.

You probably know that an ABC data form uses the ABC Window Manager Class method called `ThisWindow.Reset()` to "reset" or "synch" the update form.

Inside this reset method the form pulls together all the related pieces to be displayed and managed on the window. What happens inside this reset method depends very much on how much stuff like *fields, relations, browses and controls* you have populated on the window. This is also where things can go woefully wrong when the window is just too busy for its own good or when the data design in the dictionary is inadequate for the job.

That's another story worthy of some words of caution about the merits of streamlined window design over traditional "Clarion-Tab-Gothic" windows which tend to be overpopulated and try to do everything on one form that should be divided up amongst several forms, using well-considered work-flow-based design.

In this particular example there's not much happening on the window, only five Enrollment file fields need to be completed, that's all. No big deal. And because, as we stated earlier, due to the highly "normalized" state of the data tables, the user is left in the dark unless some fields from related tables are displayed to clarify the WHO, WHAT and WHEN aspects of the Enrollment record.

In other words:

- Who is enrolled?
- What course are they enrolled in?
- When is the class running that offers this course?

In the screen shot above, we see the "update form file tree" established on the form. This particular layout is possible because these tables are joined in the dictionary in either MANY:1 or 1:MANY relationships.

If you follow the red arrow from the circled area on the upper left, it points to some code that the ABC template writes into `ThisWindow.Reset()` that resolves the joins by fetching the related records from the joined tables. The text balloon on the right explains what's happening.

In the embed point immediately following this ABC template-embedded code after `[Priority 2800]`, we are able to update any local variables located on the window to display concatenated values from related-table fields -- unless we display the fields directly, in which case we wouldn't need to embed or concatenate anything.

You'll see what we're doing in the next image.



Rather than populating four fields from the related tables, we chose to concatenate and format these four fields into two local variables "CourseInfo" and "StudentFullName". You can see that happening, followed by a DISPLAY() in the [Priority 2800] embed point.

To avoid actually "coding" at all, we could have entered our embed using the *Formulas Editor* provided by the IDE. That drops an embed into the "Refresh Window ROUTINE After Lookups" on your behalf, though not close to the "reset" action taking place in this specific embed point. But it will work too as that routine is called when the window resets.

It's really your choice, if you prefer to use the *Formulas Editor* to assist you. Most developers can likely type faster than they can navigate that editor's user interface. And with *code assist* enabled in the standard *Embeditor* there's no real advantage using the *Formulas Editor*.



In this final image you can see a browse, and below it, the form under discussion. The two string fields pointed to by the data balloons are the local variables into which the related-file information is concatenated.

The form synchronizes with the tables when necessary and keeps these values current with the related tables whether you're in insert mode or in change mode, and you add or change a student name with the student lookup button (upper) or add/change a class name with the class lookup button (lower).

That's all we need to say on the topic of `ThisWindow.Reset()` concerning ABC Update forms and displaying related-file data. We should add that we prefer to display related data on forms in "formatted" fashion, via concatenation. On ABC forms this simple embed method (or formula insertion if you prefer) can handle that job handily.

END