Analgesia safety checklist: PDA implementation details

Part II.

Version 0.95

J.M. van Schalkwyk

February 27, 2009

Contents

1 Introduction

We continue our discussion of the database underlying our PDA-based analgesia application. In part one, we described fundamental tables which allowed us to model medical observations and intervention. In this, the second part, we discuss SQL tables which allow us to specify user menus on both the PDA and the desktop, and then implement these menus for our analgesia database.

Our objectives here are to:

1. Describe the *menu system* used to dynamically create graphical user menus on both the PDA and the desktop machine;

2. Explore our unique *scripting language*, a simple linear language used to tie together SQL statements and other commands;

3. Describe in detail the implementation of a database used to capture pain information. We describe both placement of items within user menus and the underlying scripts used to enter information.

We assume that you understand core SQL, that you know what a pixel is, and also that you have an idea how common computer interface components work, at least from a user point of view! We're talking about things like text fields, popup lists, buttons and checkboxes. Some programming skill is also assumed — for example, you should know what a stack is, and how it is pushed and popped.

Rather than having an extensive, formal discussion (or tutorial) describing how to use the scripting language, we 'show by example'. There are practical advantages to such learning on the job, although these are partially offset by the somewhat steeper learning curve.[1]

[1] We will eventually write that scripting tutorial, somewhere down the line!

2 Menu rules

Using just a handful of tables, we describe (codify) user menus, their components, and the placement of these components within the menu seen by the user. First let's look a the conventions we will use.

2.1 Rules for placement

Pixel-based graphics have a problem. If we specify the position of something on a screen, and perhaps its dimensions as well, using pixel co-ordinates, then what happens when some bright spark comes up with a screen which has more pixels on it? There is usually a cock-up — either the 'something' on the screen becomes shrunken, cramped and difficult to view (this is usually the case), or the pixel co-ordinates have to be scaled to the new pixel dimensions of the screen (which is often rather difficult to achieve, especially if the pixel values are hard-coded).[2]

Our program is confronted by the problem that it has to run, preferably without modification, on both a PDA (with a common screen resolution of either 160 by 160 or 320×320), and a desktop machine (where 160×160 looks absolutely minute). We will thus specify that all co-ordinates and dimensions we use will be specified as floating point numbers between zero and one. Such numbers represent a *portion* of the width or height of the screen; we will refrain from using pixel coordinates.[3] We will always refer our positions to the top left corner of the screen, which can be considered to occupy the position $(0, 0)$. The bottom right corner of the screen is then $(1.0, 1.0)$.[4]

We will store such floating point numbers within database tables, and then retrieve these values so we can *dynamically* create menus according to the numeric recipes we've encoded! We will do so both on desktop machines and Palm PDAs. This approach is quite different from the 'normal' way a PalmOS PDA menu is created. PalmOS menus are normally *static* — they are fixed in creation, and kept

[2] This problem is common with educational games for kids, for example. The 'solution' is often that either the game won't run on your fancy new machine with its higher screen resolution, or the program bullies the display into an older, low-resolution mode, often with dire consequences!

[3] Note however that there are some arbitrary, interesting constraints on the Palm PDA which we'll also have to take into account.

[4] 'Even better' might be to use a ratio of two numbers, as the idiosyncracies of floating point representation using finite precision numbers wouldn't then rear its ugly head. We have not followed this rather attractive path because we would then either need to create a cumbersome fraction datum type, or start playing around with pairs of integers in some other way. In addition it's rather counter-intuitive to have to find a fraction for a particular ratio, determine that this isn't quite correct, and then find another similar but slightly larger or smaller fraction. All these issues increase the likelihood of programming error. Double precision floating point numbers mitigate the problem to a degree, as does our tolerance of 'small' graphical irregularities.

in this fixed format on the PDA. We use dynamic menu creation almost exclusively, something which is difficult to achieve owing to the complexity and, dare we say it, suboptimal design of the Palm operating system. We have limited our program to PalmOS version 4.0 and above, because the dynamic components in operating systems prior to 4.0 are pretty thoroughly broken.[5]

2.2 Menu components

We will use a variety of different components to create our menus. We have set things up so that menus (made of several components) can be included as components within other menus — repeated arrangements of components don't have to be laboriously re-coded each time they are used.[6] Of course, there are practical limits to the complexity of such arrangements, and common sense should dictate even more stringent constraints on what we attempt!

We have (initially at least) been rather churlish in our allocation of menu component types. Our short list of vital components is contained in Table 1. This choice of components may seem rather arbitrary, particularly the lack of 'radio buttons', but the pushbutton can take on the same role, using less space, and is more pleasing on the eye (our eye, at least)! In this initial implementation, we have avoided use of sliders of various types, but arguments (poor ones) can be made for their later introduction.

A further pretty important component we haven't included (at present) is the bitmap. We have no bias against images, we simply haven't had time to write all of the associated code. Use of images is a priority for future development.

A popmenu does not permit the selection of multiple items. This approach keeps things simple, and constrains us to represent such multiple items as a group of pushbuttons. We are prevented from inappropriate hiding of information from the user.

[5] There are potentially effective work-arounds for earlier versions, but we won't go there, at least, not for now!

[6] Although we don't use this particular 'feature' very often!

Our term	*Perl/tk*	*PalmOS*	*Description*
label	Label	Label	Text label, cannot be edited
button	Button	buttonCtl	clickable button
checkbox	Checkbutton	checkBoxCtl	clickable checkbox with two states, ticked or not
pushbutton	[user defined]	pushButtonCtl	clickable button with two states, highlighted(on) or not. Can be grouped
text field	Entry	Field	Field in which text can be entered/edited
popmenu	Optionmenu	popupTriggerCtl	Menu of items from which one can be selected
table: monomorphic	[user defined]	[user defined]	Grouping of multiple similar items (e.g. buttons)
polymorphic	[user defined]	[user defined]	Grouping of dissimilar items in distinct columns

Table 1: Interface component types

2.3 A note on fonts and graphics

One of the most limiting characteristics of the Palm PDA was its few fonts. At present our programming is singularly bereft of font-tweaking capabilities. Ideally we should build in the ability to scale fonts to fit, just as we currently scale menu components to fit. A lot of work is needed here!

We have also completely omitted bitmap graphics from the current design. Adding such capabilities won't be particularly difficult, but we have concentrated our energies elsewhere.

3 SQL specification of menus

In this section we will describe all menu-related tables. We will start with the ITEM table which is used to represent individual *items*.

3.1 The ITEM table

```
CREATE TABLE ITEM (
  cold integer,
  iID integer,
    constraint badItemID primary key (iID),
  iType integer,
  iText varchar(64),
  iName varchar(64),
  iList varchar(1023),
  iLines decimal (2,0) default 1,
  iResponse varchar(1023),
  iInitial varchar(1023),
  iScript varchar(1023)
                );
```

As in Part I of the database, we have a cold field in each table which is exported to the PDA.

Each *item* is rather chunky, mainly due to the optional presence of lengthy associated scripts. The *iInitial* and *iResponse* scripts are respectively invoked when the item is initialised, and when it is activated by a stylus tap or mouse click. Not all items need to have initialisation or response scripts, but it's clearly a boon to be able to, for example, attach a SQL query to initialisation of an item, using the result of the query to alter the appearance or contents of the item! iScript is a little-used component at present, but has the potential to be used in communication between items![7]

Other item components are worthy of comment. The unique identifying key (iID), and text value (iText) should be self-explanatory, but what of the rest? Each item has a name (iName) of limited utility at present, other than perhaps saying what the item does; in addition multi-line items state the number of lines (iLines).

[7] Note that in some databases, for example mySQL, a *varchar* field is limited to 255 characters, so the specification of this table would have to be different.

3.1.1 Item lists

An important property of items is *iList*. This is used with popmenus, to provide a list of options, one of which can be selected. There are two ways of populating such a list. One is to hard-code the list along the lines of:

```
INSERT INTO ITEM (iID, iType, iText, iList)
          VALUES (123, 6,     'Rm', '1|1|2|2|3|3|');
```

Each item in the list is separated from the next by a 'pipe' (|) symbol. *There is also a terminal pipe, at the end of the list.* See how due to peculiarities of our coding, each item appears to be duplicated — one value is a code, and the other is the displayed representation of that code. In our example, the two are identical. More sneaky is the following:

```
INSERT INTO ITEM (iID, iType, iText, iList)
          VALUES (27, 6, 'wd', '->&ListWards');
```

What did we do in the above? The second option allows us to *invoke a script* at the time of creation of the list. The syntax is quite precise — if and only if the list value begins with `->`, then is the following script invoked, and the results of the script are used to create the list! See how we simply invoke a function, here called *ListWards* — an ampersand (&) prior to the function name tells us this is what it is, just as in Perl.

In addition, we might constrain iType to depend on another table — ITEMTYPE — as a foreign key. We won't actually implement this frilly little ITEMTYPE table, as the numeric codes for the various item types are known to our Perl and C++ programs, and don't really need to be explicitly stated. For the record, the table can be specified thus:

```
CREATE TABLE ITEMTYPE (
  cold integer,
  itID integer,
    constraint badItemtypeID primary key (itID),
  itName varchar(32)
                        );
```

The key values with their corresponding meanings are as follows:

Code	*Meaning*
1	label
2	button
3	checkbox
4	pushbutton
5	textfield
6	popmenu
7	scrollbar
8	polymorphic table
9	monomorphic table
15	bitmap
20	menu

Table 2: Menu component types

Although we will often refer to the above list during discussion, we in fact have no use for it (and ITEMTYPE) in actual SQL implementation, and will conveniently leave it out of our final database![8]

Finally, take note that because a menu is a type of item, when we come to create menus, we can insert other menus into them as items! Such menu creation is our next topic.

3.2 Menu creation

A menu is an aggregate of items. Menus are a powerful weapon, allowing us to wield several items at the same time. Let's see how this works. Here's the MENUITEMS table:

```
CREATE TABLE MENUITEMS (
  cold integer,
  miUid integer,
  miMenu integer,
    constraint badMImenu foreign key (miMenu)
       references ITEM,
  miItem integer,
    constraint badMIitem foreign key (miItem)
       references ITEM,
  miOrder integer,
    constraint badMIkey primary key (miUid),
```

[8] See how we've built bitmaps and scrollbars in, although they aren't yet implemented!

```
miX float,
miY float,
miW float,
miH float,
miPaper varchar(20),
miInk varchar(20),
miGroup decimal (2,0) default 0,
miEnabled decimal (1,0) default 1,
miInitial varchar(2048)
              );
```

We already know from the preceding section that menus too are a type of item. Now, in the above table, we associate other items with menus. miMenu tells us which menu, and miItem is the associated item. Clearly many items (including other menus) can be included as sub-components of a menu.

You can see there is one potential problem. What is the meaning of a menu which refers to *itself* as a sub-component? There is clearly potential for infinite recursion here! We use this potential flaw to our advantage, however. We make the rule that if a menu refers to itself, then it is a *top level* menu. By this, we mean that only menus which are self-referential can be displayed as individual menus on the screen. We use this self-referential property to identify such menus! It goes without saying that self-referential menus, when displayed, don't include themselves as components.

Menu descriptions obviously contain other fields. These fields include floating point representations of the size and position of the menu component (as specified in section 2.1).[9] The names of these components are pretty well self-explanatory — miX, miY, miW and miH. Menu components can be grouped (miGroup), they can be initially enabled or disabled depending on the value in miEnabled, and placement order in the menu can be specified with miOrder (smaller values in this field force earlier placement, later items potentially overlapping earlier ones). Foreground and background colour of components can be specified using miPaper and miInk, but note that at present this functionality is not enabled on the PDA.

The miInitial field is interesting. It is usually null, but I put it in as I thought that occasionally the iInitial script which is run when an item is made might need to be superseded in the context of a menu. On reflection, this is probably a bad idea, and the field should be removed.

[9] Note that due to the peculiarities of PalmOS, and particularly of the debugging environment, we've taken the line of least resistance and all of our *menus* occupy the whole PDA screen!

3.3 Table creation

We rather glossed over creation of on-screen tables. Both Perl/tk and PalmOS have the complex capability to create and display tables of similar or dissimilar items. We however define our own table-creation, which functions similarly on PDA and desktop. Central to this is an SQL table called ICOLTABLE. This table associates an item (irTBL) previously defined as a (screen) table, with another component (irItem) which is then relentlessly duplicated to constitute a column of that (screen) table. Here's the SQL:

```
CREATE TABLE ICOLTABLE (
  cold integer,
  irKey integer,
    constraint badIRowRow primary key (irKey),
  irTBL integer,
    constraint badirTBLref foreign key (irTBL)
       references ITEM,
  irItem integer,
    constraint badIrITEMref foreign key (irItem)
       references ITEM,
  irOrder decimal (2,0),
  irName varchar(64),
  irFraction float,
  irPaper varchar(20),
  irEnabled integer default 1
                );
```

See how each column (for this is in fact what we're creating) has in addition an order (irOrder) specifying where it appears as columns are created from left to right;[10] a name (irName) which is used as the title of the column; the capacity for column components to be disabled (irEnabled); and a fractional width (irFraction) which specifies what portion of the width of the table is occupied by the column. The last mentioned is necessary, because remember that a table is also an item, and as such has a screen-relative width.

A minor component is irPaper, which should allow us to alter the background colour of a particular column. We haven't implemented this ability on the PDA at present.

[10] lower values are created first

3.4 Specifying associated routines

A powerful ability of our system is that we can invoke scripts at various points. When a menu is created, and when a user selects a component with stylus/mouse, a script can be invoked. We have already seen in the preceding sections where such scripts are stored. There is a clear requirement for repeatedly used scripts to be 'encapsulated' elsewhere, and this is where *routines* come in. They are simply repetitively used scripts, kept under a single name, and invoked when needed. We even know how to invoke them within a script, as we encountered an example up above (Section 3.1.1)! We still need a table to store such routines, and here it is:

```
CREATE TABLE FUN (
  fKey integer,
    constraint badRoutineKey primary key (fKey),
  fBody varchar (2047),
  fName varchar (32),
  cold integer
              );
```

Note that because of an idiosyncracy of the C++ code on the PDA the 'cold integer' field must be provided after the other FUN fields.[11]

The name, body and key fields are self-explanatory. See how we have constrained the length of a script contained within a routine to a maximum of 2047 characters — a good length, even with fairly large embedded SQL invocations.

The name FUN is because initially I referred to these as 'functions' but they don't necessarily "take zero or more arguments off the stack and return precisely one answer". They are in fact 'relations', taking zero or more arguments off the stack and returning zero or more answers (on the stack). Hence the change of term to the slightly more neutral 'routines'.[12]

[11] This is a violation of the SQL standard, and must be fixed.

[12] I may slip from time to time — forgive me!

4 Scripting — a brief introduction

Elements within a menu such as buttons and pop-lists can be clicked on using a mouse (on the desktop machine) or a stylus (on the PDA). *Scripts* define the responses to such clicks. Our scripting language is extremely simple, and exists simply to tie menu entries into the database. The following section is a brief overview of our simple scripting language. Each command is covered in more detail in the relevant section in the document *PerlPgm.tex*, where examples of the implementation of the commands are written in the Perl programming language. For the corresponding C version, see the document *ScriptingLib.tex*.

The most powerful feature of our scripting language is the ability to script SQL queries. For example, if we wish to perform an SQL query we simply create a script containing that query, and use the QUERY keyword.

4.1 QUERY

Here's an example of QUERY in action:

```
QUERY(SELECT Person FROM BADOBS WHERE badobs = 1234)
```

We've already encountered the BADOBS table in *AnalgesiaDBpart1.tex*. There are several constraints on the above query (you might have thought of a few already!) Questions it's worth asking include:

- Where does the result of the query go?
- How do we know whether the query succeeded or failed?
- How do we take a value (for example, from a previous script) and replace, say, 1234 with that value?
- Is there anything else I should know about the QUERY instruction?

Let's address each of these in turn, but first a little note about *strings*. The statement "SELECT Person FROM BADOBS WHERE badobs = 1234" in the above QUERY is an example of a *text string*. There is another, entirely equivalent way of submitting the above statement. It is:

```
"SELECT Person FROM BADOBS WHERE badobs = 1234"->QUERY
```

See how we put the string in "double quotes" and then submit it to the QUERY statement. You'll sometimes find this variant syntax very useful.

4.1.1 Retrieving results from a QUERY

Our scripting language uses an *implicit stack*. We never refer to the stack directly, but perform operations on the stack. So if the above QUERY succeeded, it would quietly, without any fuss, put the result of the query (here, an integer representing the Person) onto the stack, where the value would be available for further use.

4.1.2 Query success?

Although SQL is often rather inscrutable in returning error codes and messages (if it does so at all) we can easily determine whether a QUERY succeeded or failed. Here's a script which does just this (and responds appropriately):

```
'QUERY(SELECT Person FROM BADOBS WHERE badobs = 1234)->
  QOK->SKIP->ALERT(Failed!)'
```

See how we've put our script within single quote marks, which allows us to do SQL-style things like:

```
UPDATE ITEM SET iResponse =
'QUERY(SELECT Person FROM BADOBS WHERE badobs = 1234)->
  QOK->SKIP->ALERT(Failed!)'
  WHERE iID = 21;
```

We can rather arbitrarily attach script *responses* to various elements in the ITEM table (described above [REF]). We've done a few other things here, so let's work carefully through the script. The following are important:

$->$ Each item in the script is separated by this strange arrow, a combination of a minus sign and a 'greater than' sign. Our scripts always work in a linear fashion, moving from left to right along the line, each item in the script being evaluated in turn!

QOK is used to determine QUERY success or failure. If the query succeeded, then one (a logical 'true') is put on the stack, otherwise a logical zero (false) is put there.

SKIP The very simple SKIP command does just what you might expect. If there is a 'true' (1) on the stack, it skips the *next* command; if there's a zero, then the skip simply doesn't occur.

ALERT displays a prompt on the screen, here the expression "Failed!".

You can work out what the above script does. If the script failed, the ALERT is displayed, otherwise the alert is simply skipped.[13]

[13]The alert reader will see how simplistic this script is, and already be asking how we respond other than simply providing an alert. This topic will be covered later!

4.1.3 Inserting a value

In our sample QUERY, it's rather limiting to 'hard code' the value 1234. We need some flexible method of pulling a value off the stack, and inserting it into the query itself. Here's the solution:

```
'#1234->QUERY(SELECT Person FROM BADOBS WHERE badobs = $[])'
```

The above script does exactly what the previous one does, but you can see that we can replace the number #1234 with any value, including a value obtained by a previous script! Several comments are in order:

- See how when we move the number out of the QUERY, we put a 'hash' (pound or tictactoe character) in front of it. This is not a strict requirement, but it does introduce us to an important concept — there are *three* different types of number in our scripting language. These are:[14]

  Integers These are always written with a hash in front of them: #1234;

  Fixed point numbers These are ordinary numbers like 1234 and 12.34.

  Floating point numbers These are at present only written thus: FLOAT(12.34)[15]

- When we wish to insert a value within a script, we put the rather clumsy character combination `$[]` at the point where the value is to be inserted. The value inserted there is the *topmost* item on the stack.

- But what if we wish to insert multiple values into a string (for example, into a QUERY statement)? Look at the following:

  ```
  "Flopsy"->"Mopsy"->"Cottontail"->"We ate $[], $[] and $[]"
  ```

  This becomes: `"We ate Flopsy, Mopsy and Cottontail"`, as you might expect. In other words, at the time the script is *parsed*, the topmost item on the stack goes into the rightmost `$[]`, and so on, working left in the string and down to deeper items on the stack.

[14] Ultimately we hope to make our number-handling completely compliant with IEEE754r. This is not a small task, and our current system is functional but rather rudimentary!

[15] In later versions of PainForm, we will probably eventually introduce a convenient shorthand, for example prefix the number with an exclamation mark to indicate it's a float!

4.1.4 QUERY limitations

There's one really vitally important limitation of QUERY, which adresses a deficiency in standard SQL. The problem is this — sometimes we wish to retrieve a *single* row from the database, but there is no standard SQL mechanism for doing so. The database engine will appear to simply plod through the whole database, even if we only want to match a single row. A query returns what it returns. We remedy this 'deficiency' by limiting our QUERY command to retrieve just a single row from SQL. Some caution is advised, because this command simply retrieves the *first* match it comes across. If you want multiple rows, then you need to use the statement described in the following section: QMANY.

Note however that there is nothing which prevents you from retrieving multiple *values* from a single *row* in the database using QUERY. The statement ...

```
QUERY(SELECT Person,Bed,Epoch FROM BADOBS WHERE badobs = 1234)
```

...is perfectly valid and will retrieve three values from the relevant row, and put three items on the stack (if they exist). It simply retrieves results from *a single row* in the database.

Note that one of the peculiarities of our current implementation of SQL on the PDA is that the comma list of SELECTed items must *not contain spaces*, so at present "SELECT Person, Bed, Epoch" will fail on the PDA (Ugh)! [CHECK ME??]

4.2 QMANY — retrieve multiple values

Consider the following script:

```
'QMANY(SELECT WARD.ward FROM WARD
WHERE WARD.ward > 0)'
```

This command QMANY is very similar to QUERY — it also queries the database and puts items on the stack. However, *multiple rows* will be retrieved. If there are 55 wards in the database, then those 55 wards will be retrieved, and the integer values of their primary keys will be placed on the stack. (It's clearly possible to crash the stack with injudicious use of this query).[16]

[16] We seriously considered having some sort of limiting argument on QMANY so that we could retrieve *exactly* N (or fewer) rows, but discarded this as desirable but too complex. Also note that our QUERY/QMANY dichotomy and other statements below are not intended to extend or supplant standard SQL, they are merely conveniences perched on top of SQL, perhaps in a somewhat clumsy way.

There's another wrinkle in the way we wrote the above statement. See how we've broken the statement across two lines. This is merely for convenience of reading — scripts are always stored without any spaces or carriage returns, and unexpected white space will usually result in an error. It is however very convenient to write scripts in a 'broken' fashion, and we will resort to this repeatedly. In our source code, we always turn scripts into single lines before we insert them into the database, using our DogWagger oneLine='yes' convention.[17]

QMANY has a minor catch for the unwary. It simply plonks all of the results retrieved on the stack, so the person performing the query will need to know how many results were retrieved for every row in the database. Ordinarily this isn't a problem. What about the case where a null result is retrieved? We have a special representation of NULL in our scripting language, in fact the script fragment:

```
'NULL->ISNULL',
```

. . . will insert a NULL value onto the stack, and then trivially test for the result using ISNULL, of course returning the value #1 (true).[18] [XREF TO SCRIPTING DESCRIPTION OF THESE: DETAIL]

We have limited our queries so that statements like "SELECT * FROM PERSON" are not supported on the PDA. In the context of our scripting language, such queries are always unhelpful and usually dangerous!

4.3 Other SQL: UPDATE, INSERT, and more!

In our scripting language, we have skeletonised SQL, down to the very basics we require. Other vital SQL commands are covered by the simple DOSQL scripting instruction. We also have rudimentary COMMIT and ROLLBACK instructions, and several necessary but 'proprietary' scripting commands.[19] Here they are:

4.3.1 DOSQL — UPDATE & INSERT

There are few wrinkles here, we simply say things like:

```
'DOSQL(UPDATE BADOBS SET boInactive=1 WHERE Person = $[])->
 DOSQL(INSERT INTO BADOBS(badobs,Bed,Epoch,Person,boFlag)
 VALUES($[],$[],$[],$[],0))'
```

[17] Look at the TEX source of this file for examples.

[18] Yes, true and false are simply the integers #1 and #0 respectively.

[19] I hasten to say that these are 'proprietary' *scripting* commands, and don't violate the spirit of SQL.

The above is an actual fragment from a script you'll encounter in this document. There are a few embarrassing little secrets about our SQL syntax on the PDA, which we still need to crack into shape. These are:

- In the SET statement within an UPDATE, there may not be spaces to the left and right of the equals sign. For example, in the above fragment "SET boInactive = 1" would fail on the PDA.
- In the INSERT statement, there are several places where spaces are similarly disallowed on the PDA. Rather arbitrarily, there may not be a space between the table name and the list of columns in parenthesis, nor may there be a space between to the left or right of the word values![20]
- In the INSERT statement, it is advisable to have the first item in parenthesis ('badobs', in the above example) as the primary key. The VALUE must of course be in a corresponding position. [CHECK ME, FIX ME]

4.3.2 COMMIT and ROLLBACK

On the PDA, once we've written to the database, as things stand we auto-commit. This is perhaps less than optimal, but we haven't yet implemented a rollback function there. The COMMIT and ROLLBACK commands should work properly in the Perl (desktop) version. They take no argument.

4.3.3 KEY

We have reasonably constrained our PDA database to work with single integer primary keys. We disallow compound primary keys. This approach works well, and suggests a further refinement — that we facilitate generation of sequential keys. Now in standard SQL there is no facility for auto-incrementing keys, so we do two things. We create a standard SQL table which contains a column for each primary key. Keys are retrieved from a single row representing our database, and the relevant field is then updated, implementing auto-incrementing keys. The second thing we do is that we create a KEY command, which, given the name of the database table, will perform the complex retrieval and incrementing of the key. The database table is called UIDS, and each 'generator' column consists of the name of the relevant database table, prefixed by the letter 'u'. So, for example, the generator column for the PERSON table is called 'uPerson'.

Here's an example of the usage of the KEY command:

[20] These irritations still need to be fixed.

```
'KEY(Medscore)->
DOSQL(INSERT INTO MEDSCORE(Epoch,msoValue,msoNature,medscore)
VALUES($[],$[],$[],$[]))'
```

In the above fragment, the remaining `$[]` values are already on the stack.

4.3.4 ME and SETME

Another 'convenience' we require is the ability to identify who is performing an action. Several of our tables need this information, encoded (of course) as an integer (primary key) value in the PERSON table. This value is supplied and stored when the user logs on, but how do we obtain it? We simply say: ME. Here's a script fragment:

```
'KEY(Epoch)->NOW->ME->DIGUP->
 DOSQL(INSERT INTO EPOCH(epoch,oMade,Person,Process)
 VALUES($[],TIMESTAMP ''$[]'',$[],$[]))'
```

From the preceding section, you know what KEY does. For now, you can more-or-less ignore the NOW and DIGUP commands (which provide a timestamp and retrieve a stored value, but more of this later)!

4.4 Menu-related commands

As we've mentioned in previous sections, a key component of our user interface is the menu which contains buttons, text fields and so forth. We have several menu-related commands, the simplest of which is MENU. For example:

```
'MENU(PATIENT)'
```

...simply takes us to the PATIENT menu, displaying that menu as specified in the database. The previous menu is stored on a special menu stack. When we wish to return from the current menu to the previous one, we simply say:

```
'MENU(#1)'
```

This command takes us back *one menu*. It is possible to pop more than one menu (go back several menus, discarding the intervening ones) by specifying a number larger than 1. MENU(#0) reloads the current menu.

4.4.1 Local variables

Within a menu we can create up to 32 special 'local variables' where we can store things. A local variable can be allocated an alphanumeric name to make its use more convenient. Look at the following snippet:

```
'NAME(id)->
 NAME(ward)->X->SET(ward)'
```

We create two names (*ward* and *id*) and then use SET to allocate a value to the *ward* variable. How then do we retrieve the value of *ward*? Thus:

```
'$[ward]'
```

Note the usage of a dollar sign followed by *square* brackets.[21]

4.4.2 Menu utilities

In order to use and modify menus and their components, the following are most useful:

ALERT Display a string on the screen;

CONFIRM Confirm an action, with YES and NO buttons; If YES is clicked, then #1 goes to the stack (true), otherwise #0.

ASK Request an input string before continuing. The string is placed on the stack (or NULL if the user cancels).

EXIT Terminates the PDA program. The user is always asked to confirm the exit. If EXIT is attempted from a menu deeper than the initial one (after log-in) then the current state is saved, so that on re-entering the PDA program, the user can resume where they left off!

ENABLED allows us to enable or disable a particular widget. At present, this command is used in the initialisation of an item. (Later we might re-institute the ability to pass a message to a menu component telling it to 'disable itself'). Submitting #0 (false) will disable a menu component.

TITLE allows us to set the title of the menu to the value of the string provided on the stack.

We also have rudimentary functions called PAPER and INK which allow us in Perl (but not yet on the PDA) to specify the colour of menu components.

[21] Although it seems logical to use a similar convention within a string, we haven't yet implemented this.

4.4.3 Fancy menu commands

You won't often need to use these commands. They are POPMENU and PUSHMENU. PUSHMENU is not actually used (and exists mainly for purposes of symmetry). POPMENU can be used to pop a menu off the menu stack. Here's an example:

```
'POPMENU(#0)->SET(stkmenu)->SET(xval)'
```

The single numeric parameter says how many menus to pop onto the stack. Menus are popped as firstly a menu name (as a string), and secondly the value of X for that menu (an integer).

Two other interesting menu commands are ROLLMENU and LINESLEFT. It sometimes happens that a table within a menu is too large to be completely displayed. When the system tries to draw the menu it records the number of lines left undisplayed, which can be retrieved using LINESLEFT. It is then possible to invoke the ROLLMENU command to re-display the menu, with the contained table shifted down to display the next set of lines. Note that ROLLMENU will push the current version of the menu to the menu stack, so if we say MENU(#1) after a successful ROLLMENU, then we return to the same menu, but with the table lines shifted back up again. Here's a snippet containing ROLLMENU which we use in displaying patients in a ward:

```
'ROLLMENU->Alert(No more!)->&ManyBack(PATIENT)'
```

If ROLLMENU fails, then processing continues as if nothing happened, but if it succeeds, then script processing ceases immediately and the new 'rolled down' version of the menu is displayed instead.

4.5 Local variables, X and V

In constructing our scripting language, we have tried to keep things simple. One aspect of this simplicity is mandating that a minimum of information is kept locally in a MENU, and that where we move from menu to menu, we do *not* pass vast swathes of information around. In fact, we are limited to passing a *single integer* between menus. All other information must be written to the database. Although this constraint seems ridiculously limiting at first, it is very powerful. Even if a script within a menu crashes (and even if the menu itself is corrupted) it is highly likely that our PDA program as a whole won't crash,[22] and that the next menu will be completely unimpaired.

[22] Other features we've engineered do help!

The single integer we pass between MENUs is simply referred to in a script as **X**. In the following example, we *assume* that the integer value in X refers to a particular patient; we use the value obtained to retrieve the patient's ID number and surname.

```
'X->&FetchIdNumber->X->&FetchSurname'
```

Don't worry about the *invocations* of the routines *FetchIdNumber* and *FetchSurname* as we'll cover this sort of behaviour in the next section. Simply see how we use the transfer variable X. (The symbol X is short for 'Xfer' or transfer).

Before we move on to examine routines, we need to find out two more things The first is how we set the value in X. We use the SETX command as shown in the following code fragment:

```
'QUERY(SELECT persdata FROM PERSDATA
 WHERE pdoSurname = ''$[]'')->QOK->
 SKIP->=Fail(No surname match!)->
 SETX'
```

You can readily work out what this does. Using a QUERY we look for any single person with the given surname, fail if there is no match, and otherwise (dangerously) set the X value to the integer key retrieved. Don't worry about what the equals sign before *Fail* means, as it's covered in the following section.

The second thing we need to do before we look at routines and flow of control, is to meet a second single-letter instruction: **V**. V is used in the peculiar context of a table displayed in a menu, be it on the PDA or in the desktop version of our program. Each and every row of a table displayed within a menu is associated with a single primary key! For example, if we are using a menu table to display information about patients on a ward, each patient will have a row in that table, and each row will be associated with the primary key for that PERSON. We can attach a script to any component of that row, and refer (in such a script) to that person using the instruction **V**. Here's an example:

```
'V->SETX->MENU(PATIENT)'
```

This script (attached to a menu of patients) gets the patient ID, sets the value of X to that value, and then moves to the PATIENT menu!

4.6 Flow of control and the stack

We've already encountered one simple instruction which allows us to 'make decisions' and move around within a script, the SKIP instruction. Computer experts

reading this code will probably be somewhat bemused by my simple SKIP, as it's the sort of instruction which was used and then abandoned very early on in the development of programming languages.

The reason why I've retained this 'trivial' instruction is simple. It *keeps things linear*. A complex source of error in programming languages is the almost invariant presence of tricky structures which determine 'flow of control'. Most modern languages have several of the following such commands: WHILE, FOR...NEXT, DO, UNTIL, IF...THEN ...ELSE, and more. These can usually be nested. In addition most modern languages have a set of complex conventions for CALLing routines, and passing values to and fro. Passing of parameters in such languages is often by value, but many allow for passing of parameters by reference, and other complex conventions.

We have just five simple commands, which are sufficient. They are:

SKIP

RETURN

REPEAT

&routine

The last item is a 'prototype' — anything beginning with an ampersand (&) is regarded as a routine, and the language interpreter looks up the routine, goes to it, and then RETURNs at the end of the routine. The alert reader will have noticed that although I said there are five commands, the above list contains just *four*. The omission is simple — you can replace the & with an equals sign (=routine) and what will happen is that there will be *no return* from the called routine.

Let's look at each of the above in turn, in more detail.

4.6.1 SKIP

SKIP is really very simple. It reads a value off the stack, and if and only if the value on the stack is the number 1, then the following instruction is skipped. The integer number 1 (#1 in a script) represents the value 'true', and only it will do! If anything other than true is on the stack, the SKIP simply won't occur. Here are two trivial examples:

```
'#1->SKIP->RETURN->ALERT(Skipped the return)'
```

In this example, the RETURN statement was skipped because we placed a true (one) on the stack. In the following example, we don't skip the ALERT statement, because a non-true value is on the stack (Here the value is zero, for 'false').

```
'#0->SKIP->ALERT(Not Skipped)'
```

See how we 'submit an argument' to the ALERT statement, by placing "Not Skipped" in parenthesis after the ALERT. Had the value on the stack been true, then both the ALERT and the "Not Skipped" would have been skipped over. We have deliberately structured things so as *not* to allow multiple arguments to be skipped over. Only a command and the associated (optional) something in parenthesis can be skipped. This limitation keeps things simple.

4.6.2 &*routine* and =*routine*

In our scripting language, named routines are stored in a table called the FUN table.[23] The script parser retrieves and runs these routines when it encounters & or = followed by the name of a function within a script. The only difference between the two options is that if you specify '=' then there is *no return* to the calling routine. You can see that this approach is very useful in combination with SKIP to implement a clean branch in code. Either you skip the invocation, or you irrevocably branch to it! SKIP is also clearly useful with & as it allows conditional calling of a routine. Best of all, the simple linear nature of such scripts removes the burden of complex nested if...then...else structures.[24]

The only way of passing parameters to and from such routines is on the stack. There is no 'clean-up' of the stack on return from a routine (but see Section 4.7 below). This approach has pros and cons. Getting rid of parameter passing and agonizing over 'by reference' versus 'by value' is a pro, in my book. Stack cleanup can be a minor irritation, on the con side.

Here's a simple example of a script which checks a date by copying it, invoking a routine called &ValiDate, and if this fails, invokes the relevant 'failure' routine.

```
'COPY->&ValiDate->SKIP->=Fail(Invalid Date: $[])->SET(dob)'
```

Here's how we create the *Fail* routine:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (176,
   'Alert($[])->FAIL',
 'Fail');
```

[23] Originally named as it stored functions, but now we're often dealing with routines which may return one, several or no values on the stack.

[24] As always, there is a cost. Often it's necessary to create 'trivial' routines where in a more complex language, a complex nested structure would have 'sufficed'!

All we do is Alert the user, pulling the offending value off the stack before we irreversibly branch to *Fail*. Note that in a lot of the following documentation, we will refer to a routine name using the outrageous convention: *Fail* rather than putting an & before it (&Fail)!

Each FUN value must clearly have a unique primary key. Ideally we should perch a 'meta-editor' above our current structure to handle such keys, but at present we manually code each function and its key.

4.6.3 RETURN

RETURN is simplicity itself. It returns flow of control to the calling script. Note that at the end of a script if there isn't a RETURN statement, there is assumed to be one, and control will automatically return to the caller. Examples of RETURN have already been discussed.

4.6.4 REPEAT/STOP

REPEAT is a little tricky. What repeat does is it repeats a given routine *forever*! Forever? Well, that's not particularly useful, is it? There is a catch. If the script interpreter encounters a STOP command, then execution of the current script will cease.[25] If that script is being REPEATed, then and only then will REPEAT stop repeating and carry on with the rest of the script in which the REPEAT command occurred.

As with all 'looping' commands, REPEAT must be treated with respect. Once you've got the hang of it, it should work well. You can see that STOP can conveniently be used with the SKIP command. Simply SKIP over a STOP if the termination condition hasn't yet been met! Here's an illustrative code fragment:

```
'MARK(#3)->
 QMANY(SELECT PROCESS.process FROM PROCESS WHERE
       PROCESS.rEnd IS NULL AND
       PROCESS.Person = $[] AND
       PROCESS.ProcType > $[] AND
       PROCESS.ProcType < $[])->
 REPEAT(&KillProc)->UNMARK'
```

We select a (potentially) long list of processes which we 'kill'; once the killing is over we proceed to the UNMARK statement (which is not strictly relevant to our current discussion, so we'll ignore it for now, as we ignore MARK). Here's a fragment from *KillProc*:

[25] Almost like a **break;** command in C.

```
'DEPTH->GREATER(#0)->SKIP->STOP'
```

DEPTH is used to check the stack depth, and if there's still something to process, we skip over the STOP statement, otherwise we terminate.

4.6.5 Guilty secrets

While I was developing the scripting language, I tried many 'experiments', and have retained a little legacy code which still needs to be removed. When using REPEAT, it's awfully convenient to 'shield' the rest of the stack from interference by marking the stack. (This approach, discussed below, is a very primitive form of 'stack clean-up'). A seemingly logical consequence was the creation of an ugly command called URZN, which stands for 'unmark and return if zero or null'. Although occasionally useful, this command is now deprecated and *should not be used*. A few instances of its use will eventually be cleaned up.

Another experiment which was much better created messaging between items within a menu. I have temporarily disabled this facility, but intend to re-create it.

4.7 Managing the stack

I believe that ideally a language should be stackless, and have conceived of such languages (also check out, for example, stackless Python). However within the constraints of the Palm environment, I rapidly came to the conclusion that the crippling liabilities of the operating system (OS) would make a stackless, heavily interrupt-driven programming language very difficult to implement without re-engineering large parts of the OS, or even writing my own OS. I therefore chose a stack-based paradigm.

Once the stack-based decision has been made, it's highly desirable to have two stacks rather than one. A dual push-down automaton is Turing-complete, but quite apart from this attractive feature, having two stacks often makes rather tricky tasks easier. We therefore have an 'implicit' stack and in addition, a second stack where we can BURY and retrieve (DIGUP) values. The simple commands available for manipulating the stack are:

COPY Copy the topmost item on the stack

DISCARD Discard the top stack item

SWOP Swop the top item and the next one down! A very useful command.

BURY Take the topmost item on the stack, and bury it on the second stack. The only way you can retrieve this value is using DIGUP.

DIGUP The opposite of BURY — retrieve a value from the second stack.

MARK Mark the stack at a specified point. Mark(#0) marks the current top of the stack, whereas MARK(#1) marks the stack one item deep to the current top, and so on. When an UNMARK occurs, all items above the mark point are discarded from the stack. Note that once a MARK has been set, all items deep to the MARKed point become inaccessible. It's as if the stack started at that point. Multiple marks can be made and then UNMARKED.

UNMARK The opposite of MARK. Once the stack is unmarked, hidden items once again become accessible (down to the previous MARK).

DEPTH The current number of items on the stack (does *not* count items below the most recent MARK).

Powerful manipulation of the stack is possible with the above commands, although admittedly BURY/DIGUP are mainly used in our code to compensate for historical inadequacies in our PDA SQL code!

4.8 Altering text

We have a number of commands devoted to manipulating text.

IN Is the string on the top of the stack contained within the next string down? Returns #1 or #0 (true or false).[26]

SPLIT Use the string on the top of the stack as the 'splitting point(s)' on which to split the next string down. The 'splitting point string' text vanishes. May produce multiple strings; if no match, leaves the deeper string unchanged and merely discards the 'splitting string' on the top of the stack.

JOIN Using the topmost string on the stack, merge *all* strings below it on the stack (down to the most recent mark), inserting the topmost string text in between all of the other strings. Useful for inserting commas or spaces, creating a single string. Note that you should probably just use `$[]` in most circumstances!

LENGTH How long is a string in 8-bit characters? We don't support UNICODE, at least for now.[27] Do not confuse with DEPTH, which is a stack instruction.

[26] Ideally we should expand this command to, perhaps, incorporating most of standard Perl regex.

[27] This is a major deficiency of the current implementation of PainForm, and must be fixed.

UPPERCASE Rather cumbersome translation of a string to uppercase (capital letters).

LOWERCASE Transform entire string to lower case. We do not at present support sentence case or other frills.[28]

CUT takes an integer on the top of the stack, and a string beneath this. The string is cut into two strings at the stated point. If the string is shorter than the cut point, it is left unchanged, and NULL is put on the top of the stack.

4.9 Arithmetic and Logical commands

We have a number of such commands, providing (at present) for basic logic and arithmetic alone. It will be easy to accommodate more advanced maths functions applied to floating point numbers, with calls to Rick Huebner's MathLib.

ISNULL Is there a NULL on the stack? NULL may have been put there, or may be the result of an SQL query, or even the result of an error in an operation! Removes the topmost value on the stack, replacing it with #0 or #1.

NULL Insert a NULL on the stack.

ISNUMBER Is the item on the stack a number? Replace the top item with the answer in a similar fashion to ISNULL.

BOOLEAN Convert the topmost item on the stack to a Boolean value. Null, zero, 'false' or 'F' all become zero, regardless of case; others default to one.

SAME Are two stack items identical? Note that 0 is not the same as #0, for example.

GREATER, LESS Greater subtracts the item on the top of the stack *from* the next (deeper) item. If the result is positive (the deeper item is greater) then #1 (true) is returned, otherwise #0. LESS is true only if the deeper item is smaller. Note the analogy to SUB (below). At present GREATER and LESS will force an error if non-integer, non-float values are submitted.

NEG Returns the negative of a number.

NOT If there's a zero on the stack, return #1. Otherwise, return #0.

[28] Should we amalgamate UPPERCASE, LOWERCASE, and more generic character-modifying abilities?

AND, OR Logical AND or OR. At present we don't have an XOR, but this is easily implemented. AND only returns #1 if two copies of #1 are present on the stack; otherwise #0 is returned. If either of the numbers isn't an integer, then an error is forced! (Use BOOLEAN to convert other types to integer logic)! It's tempting to also 'save' the error by forcing a NULL to the stack, but for now we don't.

SUB Subtract the topmost stack item from the number deep to it, and replace these two with the result.

DIV, MOD DIV divides the superficial number on the stack into the deeper one and returns the quotient; MOD is similar but returns the deeper number modulo the superficial one (i.e. it gives the remainder).

ADD, MUL ADD and MUL respectively add and multiply two numbers, replacing them with the product.

INTEGER, FLOAT INTEGER has a variety of functions (See DATE/TIME section below) but also serves to convert a floating point number (IEE754 double precision) to an integer. FLOAT converts its argument to a floating point number, and can therefore be used to turn a numeric string or fixed point number into a floating point one.

4.10 Date- and time-related commands

An important component of our SQL database is managing times, dates, and timestamps. We can convert between Gregorian and Julian dates. [We still need to do some work on daylight saving and regional time codes].

NOW Returns the current timestamp (YYYYMMDDHHMMSS). At present we have no decimal after the seconds (SS).

DATE Turns something into a Gregorian date (YYYYMMDD). If a number is supplied, it is assumed to be a Julian day number/date, and is converted to the corresponding date.

TIME Makes a time (HHMMSS). An integer or floating point number is regarded as a number of seconds (since midnight), and converted appropriately.

TIMESTAMP Similar to DATE and TIME but does the whole YYYYMMDDHHMMSS enchilada.

FLOAT Takes something and turns it into a floating point number. If that something is a date or a timestamp, then a Julian date is produced. If we submit a TIME, then the time is converted into a number of seconds, as a float.

INTEGER Turns something into an integer, if possible. Similar rules apply for dates or times submitted as arguments.

TICKS A debugging function which examines the internal clock (of the PDA; for now the Perl microtimer isn't used). It returns the number of ticks on that clock. The granularity of the PDA clock is usually about 10ms.

4.11 Caching and other functionality

SQL is complex and the complex scripting we require is time-consuming, sometimes with multiple SQL queries in one script or repeated invocation of a particular script. Optimisation is therefore important. A vital component of our optimisation is isolating (caching) data in a fairly transparent way. For example, if we're dealing with a particular person, we can conveniently and temporarily 'forget' about other people in the database, sequestering the relevant data in a cache! This caching function is at present not implemented on the desktop as it's not required there.

Consider the following snippet:

```
$[id]->CACHE(PROCESS.Person.$[])
```

Here we CACHE the data within the PERSON table specifically associated with the particular person referred to in the local variable `$[id]`. It is vitally important that when we no longer need to cache, we release this table using:

```
UNCACHE(PROCESS)
```

We can also cache a table which is dependent on another table. For example, we might cache the EPOCH table thus:

```
CACHE(PROCESS:EPOCH.Process)
```

The above convention makes it clear that the PROCESS table is the main table, that EPOCH is dependent on it, and that the linking key is *Process*.

If we cache a table dependent on another table, we should first cache the table depended on. So, for example, if we cache EPOCH and PROCESS, we first CACHE(PROCESS) and then only do we specify the above code to cache EPOCH. When we uncache these, we first uncache EPOCH, then PROCESS, i.e. we uncache in the reverse order.

When a table is cached, all of the relevant keys are retained in an internal linked list to facilitate caching of dependent tables. If there are no dependent tables due to be cached, then creation of such a list isn't necessary. To save time we can prevent creation of the list thus:

```
CACHE(PROCESS:~EPOCH.Process)
```

Inserting the tilde before EPOCH suppresses creation of a linked list when EPOCH is cached (for all of the relevant processes specified when PROCESS was cached).

4.11.1 Other functions

We won't here discuss the following commands: DEBUG, DISTINCT, FAIL, PRINT, RUN, and TEST, most of which are used infrequently or for debugging.

5 Menus for the Analgesia Database

In a previous document (PDA data capture based on a form template) we described in some detail a proposed menu system for data capture into an Analgesia Database. In the following section, we will describe the dynamic menus required to implement such a user interface. Our discussion order is more-or-less the same as that of the corresponding menus in the PDA data capture document.[29]

We do not here discuss user log-on screens, although the rudimentary capability is present within our database to implement such 'security arrangements'.[30]

[29] The current menu system differs in several significant aspects from the initial one we implemented on desktop and PDA, but as we're writing, or as is often the case, rewriting, our documentation, we will try to keep up to speed with the newer version!

[30] For current log-in capabilities, see Section 5.21.

5.1 Ward selection (900)

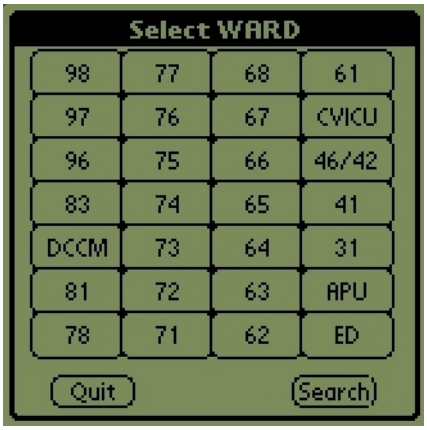


Figure 1: Ward selection menu

This menu is deceptively simple, on the surface. Deep down there is substantial complexity, with some tricky concepts to boot. The greater part of the screen is taken up by a 'monomorphic' table[31] made up of buttons, each displaying a ward name. Quit and Search buttons permit exit and searching respectively. Here's the SQL which creates the menu:

```
INSERT INTO ITEM (iID, iType, iText, iName, iLines)
  VALUES (900, 20, 'Select WARD', 'MAIN2', 1),
         (9007, 2,  '-',          'wdbtn', 1),
         (9008, 2,  'Search',      'Sbtn', 1),
         (9009, 2,  'Exit',        'Qbtn', 1),
         (9000, 2, 'New patients', 'np', 1),
         (9010, 9,  'x',         'Wd menu', 8);

UPDATE ITEM SET iInitial = '&ListWards' WHERE iID = 9010;
UPDATE ITEM SET iResponse = 'COMMIT->EXIT'  WHERE iID = 9009;
UPDATE ITEM SET iResponse = 'MENU(SEARCH)' WHERE iID = 9008;
UPDATE ITEM SET iInitial = 'COPY->$[activeW]->SWOP->IN->NOT->SKIP->TOGGLE' WHERE i
UPDATE ITEM SET iResponse = 'V->SETX->MENU(PATIENT)' WHERE iID = 9007;
UPDATE ITEM SET iResponse = '#1->SETX->MENU(NEWPTS)' WHERE iID = 9000;
--- Item 9000 is a later addition. See the New Patients menu!
UPDATE ITEM SET iInitial = '$[activeW]->"new"->IN->NOT->SKIP->TOGGLE' WHERE iID =

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
```

[31] 'Monomorphic refers to the fact that the table is made up of several columns of identical items, unlike a polymorphic table which has several rows potentially made up of dissimilar items.

```
  VALUES (901, 900, 900,  0, 0.001, 0.001, 0.999, 0.999, 0),
         (992, 900, 9008, 2, 0.750, 0.910, 0.200, 0.080, 0),
         (993, 900, 9009, 1, 0.050, 0.910, 0.200, 0.080, 0),
         (995, 900, 9000, 4, 0.31,  0.910, 0.38,  0.080, 0),
         (994, 900, 9010, 3, 0.015, 0.030, 0.970, 0.881, 0);

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName,
                       irFraction)
  VALUES (1, 9010, 9007, 1, 'rowbtn', 0.25);
```

Several features are evident — the sparse nature of the data elements (for example, there is only one iInitial script); that we *do* need to start exploring how scripts work; and the complex relationships between menu components. Recall the component types from table 2 — 2 is a button, 9 a monomorphic table, and 20 a menu.

Let's look in more detail. What we do above is create five items, with fairly arbitrary ID numbers (7–10, and 900; we will generally use larger ID numbers for items which are also menus, as a sort of aide mémoire). We next group the items into the menu (making the menu self-referential, as described above in section 3.2).[32] Finally, we create a monomorphic table to contain the ward names, with each column of buttons taking up 25% of the table width.

Where items have associated scripts, rather than including these scripts in the initial INSERT statement, we add an UPDATE statement. Let's look at the associated scripting. The `EXIT` and `MENU(SEARCH)` statements should be self-explanatory. The *ListWards* routine we'll explore in a moment, but what about the fragment `V->SETX`? It should be clear that the sequence `->` is used to lead from one script component to the next. but what are the two components? We have taken a rather abrupt leap into scripting, because `V->SETX` contains several complex ideas. These are:

- In a monomorphic table, we will always map each copy of the single component to a unique ID of something within our database! The *v*alue of the ID *v*ariable associated with such a component is referred to as 'V'.
- When we move from one menu to the next, we only ever pass *one* datum to the following menu!! This datum is referred to as 'X'.[33]

The former idea means that we can trigger a unique response to a click on each button (here, go to a particular ward); the latter makes for extremely robust programming, and constrains us to careful, logical menu design.

[32] You can see that creation of menus would be facilitated by a new level of abstraction where we didn't have to painstakingly write the SQL! This program is planned but no such animal yet exists in our suite of programs.

[33] V is for variable, X is for 'transfer' ie. Xfer!

By default, the datum passed to the next menu is the datum received by the current one (which begs the question "What is the value of this datum in this, the first menu?"[34]) but in our particular circumstance, we wish to pass the ID of the *ward* we will 'visit'. Fortunately, we can access this ID — we just refer to it as V. We set the new value of X that we will transfer to the WARD menu by saying SETX. We will sometimes refer to X as the *subject* of the menu. And that's about it, apart from the minor matter of *ListWards*.

5.1.1 Obtaining a ward list

Here's the routine *ListWards*, spread over several lines for easy reading. It obtains the IDs of all wards. As is now the case for all poplists, we create *pairs* of variables, the ID of the ward associated with the actual ward number!

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (5,
   'QMANY(SELECT ward,swrdText FROM WARD
   WHERE ward > 1 AND cold IS NULL)',
   'ListWards');
```

See how we embed an SQL statement in a script! But take careful note — we have chosen to add a further layer of abstraction between ourselves and ODBC. To exclude the staging 'new' ward, have $<> 1$ in place of > 0. By checking for the 'cold' flag, we allow unused wards to be hidden from sight. We will script our SQL in three quite distinct ways. The options are:

QUERY — retrieve a *single item* using an SQL query;

QMANY — retrieve *multiple* items;

DOSQL — execute an SQL statement, but retrieve *nothing*, for example, perform an UPDATE or INSERT.

You can see the power and sense of scripting our SQL thus. This approach can even contribute to query optimisation. In the above statement, we need a whole list of wards, so we use QMANY to obtain that list. The exclusion of a ward ID of 1 is because in our listing of wards, we will also create an 'new' or 'unknown' ward, to allow for the case where we know the person is somewhere, but don't know where they are.[35]

[34] 'The ID of the user currently logged on!'

[35] Of course, we might wish to view this 'ward', in which case the SQL could be modified accordingly.

Actually, the above statement is inelegant, because for each ward we obtain not only the *ID* of the ward, but also its name. This seems ugly, but works well for creation of a monomorphic table — it makes sense to simply provide both values routinely, rather than simply providing just the ID, and then have the monomorphic script routines *look up* the text name corresponding to each ID![36]

5.1.2 An experimental routine

Let's consider a (tentative) routine to pick out just the active wards. This routine isn't actually used, as we take a different approach!

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (6,
   '#0->SET(useW)->
   MARK(#0)->BURY(#0)->
     QMANY(SELECT Bed FROM BADOBS WHERE boFlag <> 0
      AND boInactive IS NULL
      AND cold IS NULL ORDER BY Bed)->
     REPEAT(&SetActive)->REPEAT(&Retrieve)->
    JOIN( )->Alert(Flagged patients are in $[])->
   UNMARK',
   'ShowActiveWards');
```

In the following we might use the experimental 'send' routine to send a message (just a blank) to the widget named according to the number of the ward!

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (7,
   'DEPTH->GREATER(#0)->SKIP->STOP->
    DIV(#10000)->INTEGER->SET(useW)->
    DIGUP->COPY->BURY->
    $[useW]->SAME->NOT->SKIP->RETURN->$[useW]->BURY',
   'SetActive');
```

5.1.3 Listing rooms within a ward

Here's a similar routine to list rooms, given a particular ward. As usual, we list couplets of ID and text for each room.

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (249,
    'QMANY(SELECT room,srmText FROM ROOM WHERE Ward = $[])',
  'ListRooms');
```

[36] We use an identical approach for creating poplists, showing the power of this associative method!

5.2 Patient selection within a ward (920)

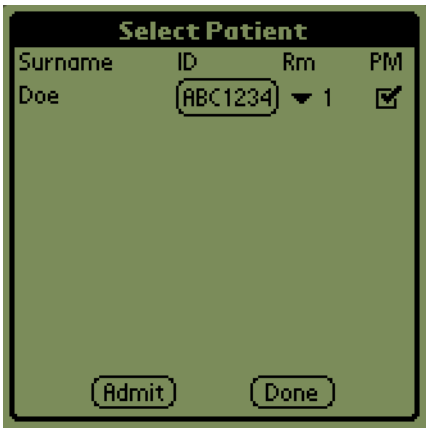


Figure 2: Patient selection menu

In this menu we demonstrate use of our other complex component — the polymorphic table! This component (which contains patient information, and allows access to individual patient details), and two buttons make up the whole menu. The two buttons are pretty well self-explanatory, Done to return to the previous menu, and Admit to admit a new patient to the analgesia service. Components of the table will also allow us to demonstrate labels, popmenus and checkboxes.

An important question is "Which patient details are sufficient?" We are constrained to a degree by the resolution of the PDA screen[37]. Here are the fields we have chosen:

- Surname
- ID
- Room
- PM[38]

The ID refers to the unique hospital number of the patient,[39] room and surname are self-explanatory,[40] and 'PM' is used to indicate at a glance whether the patient is 'special', using a checkbox. Here's the SQL:

[37] and by a desire not to make any screen too busy

[38] We have recently amended this to a more generic symbol: [!] — the exclamation mark signals that the patient is 'flagged' for one of several reasons, including problems, PM review, and 'not yet seen today'.

[39] In New Zealand, generally the 'NHI', or National Health Index number.

[40] Noting that we do not mangle compound surnames

```
INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (920, 20, 'Select Patient', 'PATIENT');

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (9920, 1,     's',    'Sn' ),
                 (9921, 2,     'i',    'ID' ),
                 (9923, 3,     'p',    'PM' ),
                 (9925, 2,     'Back', 'Bk' ),
                 (9926, 2,     'More', 'Bk' ),
                 (9927, 2,     'Admit','Adm');

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (9922, 6,  'r',  'Room',
  '->X->&ListRooms');

INSERT INTO ITEM (iID, iType, iText, iName, iLines)
  VALUES (1240,8,'[No patient found for this ward]','PtTbl',8);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (921 ,920, 920, 0, 0.001, 0.001, 0.990, 0.990),
         (924, 920, 1240,  1, 0.001, 0.001, 0.999, 0.850),
         (9925, 920, 9925,  2, 0.05, 0.900, 0.200, 0.080),
         (925, 920, 9926,99, 0.75, 0.900, 0.200, 0.080),
         (923, 920, 9927,3, 0.400, 0.900, 0.200, 0.080);

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
                       irName, irFraction, irEnabled)
  VALUES (9923, 1240, 9923, 4, '-!-',      0.11, 0),
         (9922, 1240, 9922, 3, 'Rm',      0.20, 1),
         (9921, 1240, 9921, 2, 'ID',      0.299, 1),
         (9920, 1240, 9920, 1, 'Surname', 0.40, 1);
```

Explanation is in order. First, we create the PATIENT menu, then the items which populate it. These items are the two buttons, and the polymorphic menu together with its components — a label for the surname, a button containing the ID, a checkbox for the '-!-' alert value, and a popmenu for the room within the ward.

We used to hard-code the list of items in the popmenu.[41] There are a few other frills. We use the iText field of the polymorphic menu to provide a default message ('No patient found...');[42] as usual the names are arbitrary conveniences of little worth; and the iText values of some items seem rather arbitrary and meaningless — they are. Also see how the '-!-' checkbox is disabled, preventing the casual user from here altering it!

[41] Now we dynamically create this list.

[42] Display of this message is still to be implemented on the PDA.

On entering the menu we also create the local variable $id (used later) and the $ward variable which records the ID of the current ward, obtaining this value from the transfer variable X. The $ward variable is for the use of *EnterDetailMenu* which in turn might invoke *ReadmitPatient*:

```
UPDATE ITEM SET iInitial =
    'NAME(id)->
     NAME(ward)->X->SET(ward)->
     X->TITLE(Ward $[])'
    WHERE iID = 920;
```

We still need to attach functionality to the various buttons and other components. Let's do so:

```
UPDATE ITEM SET iInitial =
  'LINESLEFT->BOOLEAN->SKIP->STOP->RETURN'
  WHERE iID = 9926;
-- [test and fix the above]
UPDATE ITEM SET iResponse =
  'ROLLMENU->Alert(No more!)->&ManyBack(PATIENT)'
  WHERE iID = 9926;
UPDATE ITEM SET iResponse = 'MENU(1)'
  WHERE iID = 9925;
UPDATE ITEM SET iResponse = 'MENU(ADMIT)'
 WHERE iID = 9927;

-- THE FOLLOWING IS STRICTLY FOR DEBUGGING!
UPDATE ITEM SET iInitial = ''
  WHERE iID = 9927;
```

A particularly important initialisation is the following:

```
UPDATE ITEM SET iInitial = '&GetBadobs4Ward'
  WHERE iID = 1240;
```

We initialise the polymorphic table (1240) with bed observations peculiar to current patients in this ward. The relevant routine (GetBadobs4Ward) is described below.

```
UPDATE ITEM SET iInitial  =
  'V->QUERY(SELECT Person FROM BADOBS WHERE badobs = $[])->&FetchSurname'
  WHERE iID = 9920;
UPDATE ITEM SET iInitial  =
  'V->QUERY(SELECT Person FROM BADOBS WHERE badobs = $[])->&FetchIdNumber'
  WHERE iID =9921;
-- must also create new epoch!
```

```
UPDATE ITEM SET iResponse =
  'V->QUERY(SELECT Person FROM BADOBS WHERE badobs = $[])->COPY->SETX->
  CACHE(PROCESS.Person.$[])->
  CACHE(PROCESS:~EPOCH.Process)->MENU(INTRO)'
  WHERE iID = 9921;

UPDATE ITEM SET iInitial  = 'V->&GetAlert'
  WHERE iID = 9923;
UPDATE ITEM SET iInitial  = 'V->&GetPatientRoom'
  WHERE iID =9922;
UPDATE ITEM SET iResponse =
  'V->QUERY(SELECT Person FROM BADOBS WHERE badobs = $[])->&SetNewRoom'
  WHERE iID = 9922;
```

Already we're getting used to the scripting — From the previous menu, we understand what V means, and we can see that a lot of the scripting is simply invocation of various *routines*. As for a monomorphic table, we can use a script to initialise values of elements.

Here's a recently introduced curve ball — *ManyBack*. This routine repeatedly pops menu data using POPMENU.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (243,
    'NAME(mnu)->NAME(stkmenu)->NAME(xval)->
     SET(mnu)->
     REPEAT(&ManySub)->
     $[xval]->SETX->$[stkmenu]->MENU',
 'ManyBack');
```

We repeatedly pop menus until one *not* identical to the submitted name is encountered. At this point we push this menu name back to the stack and reload the menu!

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (244,
    'POPMENU(#0)->SET(stkmenu)->SET(xval)->
    $[mnu]->$[stkmenu]->SAME->
    SKIP->STOP->RETURN',
 'ManySub');
```

The *EnterDetailMenu* routine merits careful scrutiny. It accepts the patient ID in the transfer variable X. First it finds the general observation process for that patient[43] and creates a *new* epoch for that process. Finally, the routine enters the DETAILS menu.

[43] Which must exist, as the patient has been admitted

A **recent test** (4/2007) is the introduction of SQL caching while dealing with a particular patient. We used to submit the patient ID to the CACHE function in this routine, but have now moved it out to MENU 970 (INTRO).

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (162,
        'X->
        QUERY(SELECT PROCESS.process FROM PROCESS
          WHERE PROCESS.Person = $[] AND
          PROCESS.ProcType = 1 AND
          PROCESS.rEnd IS NULL)->
        QOK->SKIP->=ReadmitPatient->
        BURY->
        KEY(Epoch)->NOW->ME->DIGUP->
        DOSQL(INSERT INTO EPOCH(epoch,oMade,Person,Process)
         VALUES($[],TIMESTAMP ''$[]'',$[],$[]))->
         MENU(DETAILS)',

        'EnterDetailMenu');
```

If we cannot find an active type 1 process then the patient isn't currently admitted[44] and we first have to readmit them! This routine is discussed in *ReadmitPatient*:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (239,
    'CONFIRM(Re-admit patient?)->SKIP->MENU(0)->
    #3->&ProcAndEpoch->BURY->
     "Badobs"->KEY->
     $[ward]->#10000->MUL->
    DIGUP->X->
      COPY->
      DOSQL(UPDATE BADOBS SET boInactive=1 WHERE Person = $[])->
    DOSQL(INSERT INTO BADOBS(badobs,Bed,Epoch,Person,boFlag)
    VALUES($[],$[],$[],$[],0))->
    #1->&ProcAndEpoch->
    $[id]->SetX->
    MENU(DETAILS)',
   'ReadmitPatient');
```

In the above, which can only be invoked from the INTRO menu, X is the patient in question. The SetX is for the invocation of ReadmitPatient within a ward menu.

The precautionary UPDATE BADOBS might be unnecessary if by default when we discharge a patient, we set this flag anyway!

[44] Technically, for *admission* we should look for the type 3 process. [CHECK THIS]

Given the patient ID, we create admission processes in a similar fashion to the initial admission process *CreateAdmission*. We also make a generic observation and a BED observation before moving to the DETAILS menu.

There are several catches: firstly, we might have no current ward, and secondly the clumsy function *ProcAndEpoch* also requires the local variable `id` to contain the patient ID.

We've already addressed the need for $[id], and the invoking menu ultimately always specifies a value for $ward, even if this is only 1 to represent 'ward unknown'. Associated with this is a room coded as 100 and a generic BED coded as 10000, which we use here.[45]

5.2.1 Selection routines

We still need to examine the scripts themselves — no mean task — so let's look at them one by one. For notational convenience, we will separate out the body of the routine, as in the following example.

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(100,

'X->MUL(#10000)->ADD(#10000)->
 X->MUL(#10000)->SUB(#1)->
 QMANY(SELECT badobs FROM BADOBS WHERE
   cold IS NULL AND
   boInactive IS NULL
   AND Bed < $[] AND Bed > $[] ORDER BY Bed)->
  QOK->SKIP->STOP->RETURN',

'GetBadobs4Ward');
```

Given the ward ID as X, we find all patients for this ward. Instead of returning patient IDs, we sneakily return the *key* of the actual observation, which is more useful overall! We work through the following sequence:

1. Create minimum and maximum BED IDs from the ward ID. To do so, multiply by 10000, copy, and either subtract 1 or add 10000.[46]

2. Get active beds for this ward from the ugly BADOBS table. Active beds are fairly complex in that there are two possible flags *cold* and *boInactive*![47]

[45] As GetPatientWard still works in obtaining the most recent ward occupied by a patient, the alternative to allocating an unknown ward is to use this ward, but we'd rather not.

[46] This relies on our convention that the fixed ID of a BED is a number multiplied by 10 000, that of a room is multiplied by 100, and a ward is simply a number from 1–100.

[47] This is largely legacy stuff, as all we need is *cold* but we initially designed in boInactive. We

Retrieve

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(102,
 'DIGUP->COPY->#0->SAME->SKIP->RETURN->DISCARD->STOP',
'Retrieve');
```

[dig up value, if not zero, return, otherwise skip, discard zero value and stop] [THIS FX IS LIKEWISE CRAP AND WE SHOULD FIX UP ERROR HANDLING TO FORCE 'STOP'] [THEN CAN GET RID OF COMPLEX COMPARISON] [AT PRESENT WE ONLY STOP IF ZERO DUG UP IE AT BOTTOM OF DIGUP] HMM ONE WAY OF DOING THIS WHICH MAKES SOME SENSE IS FOR URZN TO FORCE A STOP CONDITION!!

Every time we invoke *Retrieve*, we dig up a value out of storage (previously stored there using BURY), until there's nothing more, at which point REPEAT fails.

Next, let's look at retrieval of various column values in our polymorphic table — surname, patient ID number, room, and whether the patient is for 'PM review'.

FetchSurname

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(103,

'QMANY(SELECT MAX(PERSDATA.persdata) FROM PERSDATA
  WHERE PERSDATA.pdoPerson = $[])->
QUERY(SELECT PERSDATA.pdoSurname FROM PERSDATA
  WHERE PERSDATA.persdata = $[])',

'FetchSurname');
```

The script is clumsy but self-explanatory: first, get the most recent personal data epoch containing a non-null value for the surname, then retrieve this value. Retrieval of the ID number is almost identical:

FetchIdNumber

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(104,

'QMANY(SELECT MAX(PERSDATA.persdata) FROM PERSDATA
  WHERE PERSDATA.pdoPerson = $[])->
   QOK->SKIP->RETURN(?)->
QUERY(SELECT PERSDATA.pdoHospNo FROM PERSDATA
```

might profitably remove boInactive completely, provided we don't want to provide a history of bed movements on the PDA, as any cold item will not be moved back to the PDA. I think it's best to allow this functionality, so have left boInactive in.

```
  WHERE PERSDATA.persdata = $[])',

'FetchIdNumber');
```

So similar, in fact that we don't need to explain the script! Let's rather look at retrieving the current room for the patient:

GetPatientRoom

Given the record key in BADOBS, trivially pull out the room.

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(106,

'QUERY(SELECT Bed FROM BADOBS WHERE badobs = $[])->
   QOK->SKIP->RETURN(?)->
  #100->DIV->INTEGER->
QUERY(SELECT ROOM.srmText FROM ROOM
 WHERE ROOM.room = $[])',

'GetPatientRoom');
```

There is another routine we have to examine in relation to the room number. If the user *alters* the room, then we will have to respond by updating the database.

SetNewRoom

Here we submit the ID of the patient on the stack, with the ID of the room below this. We copy the patient ID and bury the copy. First we find the relevant admission process (code 3) and create a new epoch on this (INSERT INTO EPOCH..).

We then bury the ID of the epoch *and* the room ID, create a new key for BADOBS, and in sequence dig up the room ID which we multiply by 100, epoch, and the patient ID! We can now populate a BEDOBS row. We multiply by 100 to obtain a generic BED ID for that room, as at present we aren't coding at BED level!

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(107,

'COPY->
 BURY->
  QUERY(SELECT PROCESS.process FROM PROCESS
    WHERE PROCESS.Person = $[]
    AND PROCESS.ProcType = 3
    AND PROCESS.rEnd IS NULL)->
    QOK->SKIP->=Fail(Not admitted!)->
  BURY->KEY(Epoch)->COPY->NOW->ME->DIGUP->
```

```
  DOSQL(INSERT INTO EPOCH(epoch,oMade,Person,Process)
  VALUES($[],TIMESTAMP ''$[]'',$[],$[]))->
  BURY->
   BURY->
    KEY(Badobs)->#100->DIGUP->MUL->DIGUP->DIGUP->
 COPY->COPY->BURY->
  QUERY(SELECT boFlag FROM BADOBS WHERE boInactive IS NULL AND
    Person = $[])->
  DIGUP->
 DOSQL(UPDATE BADOBS SET boInactive=1 WHERE Person = $[])->
 DOSQL(INSERT INTO BADOBS(badobs,Bed,Epoch,Person,boFlag)
    VALUES($[],$[],$[],$[],$[]))',
'SetNewRoom');
```

Until 2007-11-13 the above routine did not preserve boFlag, which was irritating. We now do.

The UPDATE BADOBS statement allows us to keep just one active BED record, which simplifies searches immensely.

Finally, let's find out how we detect whether evening observation is required for a patient.

GetPmFlag

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(108,

  'QUERY(SELECT PROCESS.process FROM PROCESS
    WHERE PROCESS.Person = $[]
    AND PROCESS.ProcType = 1100
    AND PROCESS.rEnd IS NULL)->
  QOK->SKIP->RETURN(#0)->RETURN',
'GetPmFlag');
```

We have a separate 'evening observation process'. In the above, we check for the existence of such a non-terminated process for this patient. If the SQL fails, we return zero (already on the stack); otherwise we swop the top two items on the stack, then discard the top one, and return the (nonzero) process code.

5.3 Patient admission (905)

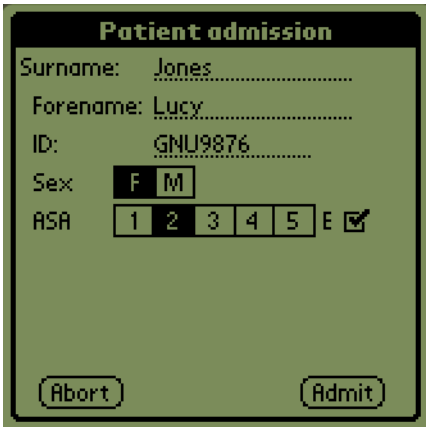


Figure 3: Patient admission menu

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (905, 20, 'Patient admission', 'ADMIT');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (5, 905, 905, 0,
          0.001, 0.001, 0.990, 0.990, 0);
```

The admission menu (ID 905) is reasonably straightforward, but is complicated by the necessity to record several data items *before* admission. There are several ways we might do this, but we choose to allocate local variables to each datum, and then, when we admit, we pull data items out of the local variables. [48]

The local variables are:

- surname
- forename
- hospitalnumber
- sex (gender)
- ASA rating and ASA E rating

Here's the simple script to create the temporary variables. By convention, we don't have to initialise them to null, as this is automatically performed.

[48] Better than temporarily writing to the database, and then rolling back if we cancel!

```
UPDATE ITEM SET iInitial = 'NAME(surname)->
   NAME(forename)->NAME(hospitalnumber)->
   NAME(sex)->NAME(ASA)->NAME(ASAe)->NAME(wt)->
   NAME(dob)->
   NAME(ward)->X->SET(ward)->
   NAME(EpL)->
   NAME(id)' WHERE iID = 905;
```

Here's the population of the menu with various items: the abort button (code 80), surname (code 82), forename (84), hospital number (NHI, code 86), gender (two pushbuttons, codes 87 and 88), ASA rating (codes 90–94), and ASA E score (code 95). The variable 'id' isn't used immediately, but is used when we actually record the patient admission in the database!

At present we do not check for existence of a duplicate NHI, but we should do so to prevent duplicate entries!

In the above we've also added a $ward variable. This is of general utility, and allows invoked routines to be independent of their former need to use X as the ward ID.

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (80, 2,  'Abort',  'Abrt', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (106, 905, 80, 23,
          0.070, 0.900, 0.200, 0.08, 6, 'red', 'white');
UPDATE ITEM SET iResponse = 'MENU(1)' WHERE iID = 80;


INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (81, 1,  'Surname: ',  'asur', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (107, 905, 81, 2,
          0.050, 0.150, 0.200, 0.08, 0);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (82, 10,  '',  'SurTxt', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (108, 905, 82, 3,
          0.350, 0.150, 0.500, 0.08, 0);
UPDATE ITEM SET iResponse = '&FixSQL->SET(surname)' WHERE iID = 82;


INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (83, 1,  'Forename: ',  'Pt1st', '',  1);
```

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (109, 905, 83, 4,
          0.050, 0.250, 0.200, 0.08, 0);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (84, 10,  '',  'Txt1st', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (110, 905, 84, 5,
          0.350, 0.250, 0.500, 0.08, 0);
UPDATE ITEM SET iResponse = '&FixSQL->SET(forename)' WHERE iID = 84;



INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (85, 1,  'NHI: ',  'HNo', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (111, 905, 85, 1,
          0.050, 0.050, 0.100, 0.08, 0);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (86, 10,  '',  'TxNhi', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (112, 905, 86, 2,
          0.350, 0.050, 0.400, 0.08, 0);
```

We also add optional information: the weight and birth date:

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (98, 1,  'Optional:  Weight (kg)',  'wti');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (98, 905, 98, 17,
          0.05, 0.550, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (99, 14,  '',  'wtx', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (99, 905, 99, 17,
          0.69, 0.550, 0.18, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (198, 1,  'Birth date:',  'wti');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
```

```
  VALUES (198, 905, 198, 19,
          0.25, 0.650, 0.25, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (199, 10,  '',  'dob', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (199, 905, 199, 20,
          0.54, 0.650, 0.33, 0.08, 0);
-- we don't use date picker here as is clumsy

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (197, 1,  '(yyyy-mm-dd)',  'wti');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (197, 905, 197, 21,
          0.53, 0.73, 0.33, 0.07, 0);
```

Here's the routine to validate and set the weight value:

```
UPDATE ITEM SET iResponse =
  'FLOAT->COPY->ISNULL->NOT->SKIP->=FailAndReload(Not a number)->
  FLOAT(1000)->MUL->INTEGER->
  SET(wt)'
  WHERE iID = 99;
```

For now, we read the number as a float, multiply by 1000 and then convert to an integer weight in grams. For the date of birth we need

```
UPDATE ITEM SET iResponse =
  '&Kiwidate->
  COPY->&ValiDate->SKIP->=Fail(Invalid Date: $[])->SET(dob)'
  WHERE iID = 199;
```

We introduce a small routine to turn around a New Zealand style date (DD-MM-YYYY) into an international one (YYYY-MM-DD). It has two subsidiary routines, and returns NULL on failure, otherwise a formatted date. It even allows single digit day and month, but the year must be 4 digits.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (282,
  'MARK(#1)->
  SPLIT(-)->
  DEPTH->SAME(#3)->SKIP->SPLIT(/)->
  DEPTH->SAME(#3)->SKIP->=IsBad->
  COPY->LENGTH->SAME(#4)->NOT->SKIP->=ReverseDate->
```

```
  COPY->LENGTH->GREATER(#1)->SKIP->"0$[]"->BURY->
  COPY->LENGTH->GREATER(#1)->SKIP->"0$[]"->BURY->
  COPY->LENGTH->SAME(#4)->SKIP->=IsBad->BURY->
UNMARK->DIGUP->DIGUP->DIGUP->"$[]-$[]-$[]"',
 'Kiwidate');

INSERT INTO FUN (fKey, fBody, fName)
    VALUES (283,
    'SWOP->BURY->SWOP->DIGUP->SWOP->
COPY->LENGTH->SAME(#4)->NOT->SKIP->=IsBad->
"$[]-$[]-$[]"->BURY->UNMARK->DIGUP->&Kiwidate',
 'ReverseDate');

INSERT INTO FUN (fKey, fBody, fName)
    VALUES (284,
    'UNMARK->NULL',
 'IsBad');
```

Here's the date validation routine. We utilise the fact that if we convert a date to and from a Julian date using FLOAT, the value should be the same!

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(263,
  'TIMESTAMP->
   COPY->ISNULL->NOT->SKIP->RETURN->
   COPY->
   FLOAT->TIMESTAMP->SAME',
'ValiDate');
```

In the above we should probably trim spaces at the start and the end of the date, as otherwise some user will become unhappy sometime.We perform the initial TIMESTAMP to allow for either a date or a timestamp.[49]

The hospital number is a little more complex, as we have to check that no existing patient on the PDA has this 'NHI':

```
UPDATE ITEM SET iResponse =
  'NULL->SET(hospitalnumber)->
   COPY->&GoodNhi->BOOLEAN->SKIP->=Fail(Invalid NHI)->
   UPPERCASE->COPY->SET(hospitalnumber)->
   QUERY(SELECT PERSDATA.pdoPerson FROM PERSDATA
   WHERE PERSDATA.pdoHospNo = ''$[]'')->
   QOK->SKIP->RETURN->
   COPY->SET(id)->
   QUERY(SELECT BADOBS.Bed
    FROM BADOBS WHERE BADOBS.Person = $[] AND BADOBS.boInactive IS NULL
```

[49]The TIMESTAMP function will convert a date to a full timestamp, but leave a timestamp unaltered.

```
    AND BADOBS.cold IS NOT NULL)->
   QOK->SKIP->=ReadmitPatient->
  #10000->DIV->INTEGER->
  QUERY(SELECT WARD.swrdText FROM WARD
    WHERE WARD.ward = $[])->
   Alert(Patient is in Ward $[]!)->
   POPMENU(#0)->DISCARD->DISCARD->MENU(0)'
  WHERE iID = 86;
```

In the above we have stolen the 'QUERY(SELECT BADOBS.Bed' code from *GetMyWard*. Formerly we simply entered the patient menu, but owing to the Palm PDA's odd handling of events[50] there is the potential to crash horribly if we enter an existing NHI and then click on [Admit]! So we now display the ward, and nothing more.

If the NHI is not found, we return with the NHI in `hospitalnumber`. The code is similar to that for the search button code.

Here's a routine to validate a New Zealand NHI (3 letters followed by 4 numbers):[51]

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (285,
  'MARK(#1)->
  COPY->LENGTH->SAME(#7)->SKIP->=IsBad->
  UPPERCASE->""->SPLIT->
  &IsNumb->SKIP->=IsBad->
  &IsNumb->SKIP->=IsBad->
  &IsNumb->SKIP->=IsBad->
  &IsNumb->SKIP->=IsBad->
  &IsAlpha->SKIP->=IsBad->
  &IsAlpha->SKIP->=IsBad->
  &IsAlpha->SKIP->=IsBad->
  UNMARK->#1',
 'GoodNhi');
```

IsNumb accepts a numeric string of 1 character, and returns #1 or NULL.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (286,
    'COPY->Greater(/)->SKIP->RETURN(#0)->LESS(:)',
 'IsNumb');
```

IsAlpha is similar to IsNumb but checks for an uppercase alpha.

[50] Or, at least, our use of this handling!

[51] We really must write those regex routines!

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (287,
    'COPY->GREATER(@)->SKIP->RETURN(#0)->LESS([)',
 'IsAlpha');
```

The POPMENU instruction removes the 'Admit' menu from the menu stack; we also must set the $id variable, and clumsily set X, the standard transfer variable between menus, to the internal patient ID. The $ward variable (used by *Readmit-Patient*) has already been set in the menu containing the button with ID 86.

Gender buttons:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (87, 4,  'F',  'Female', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (88, 4,  'M',  'Male', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (114, 905, 87, 8,
          0.250, 0.350, 0.100, 0.08, 1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (115, 905, 88, 9,
          0.350, 0.350, 0.100, 0.08, 1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (97, 1,  'Sex',  'MfLbl', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (116, 905, 97, 7,
          0.050, 0.350, 0.100, 0.08, 0);
--- Pushbuttons for gender M|F
UPDATE ITEM SET iResponse = 'DISCARD->#2->SET(sex)'
    WHERE iID = 88;
UPDATE ITEM SET iResponse = 'DISCARD->#1->SET(sex)'
    WHERE iID = 87;
```

ASA scoring:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (89, 1,  'ASA',  'ASA score', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (90, 4,  '1',  'ASA1', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (91, 4,  '2',  'ASA2', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (92, 4,  '3',  'ASA3', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
```

```
          VALUES (93, 4,  '4',  'ASA4', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (94, 4,  '5',  'ASA5', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (117, 905, 89, 10,
          0.050, 0.450, 0.050, 0.08, 0);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (118, 905, 90, 11,
          0.250, 0.450, 0.100, 0.08, 2);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (119, 905, 91, 12,
          0.350, 0.450, 0.100, 0.08, 2);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (120, 905, 92, 13,
          0.450, 0.450, 0.100, 0.08, 2);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (121, 905, 93, 14,
          0.550, 0.450, 0.100, 0.08, 2);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (122, 905, 94, 15,
          0.650, 0.450, 0.100, 0.08, 2);

--- code for ASA 1..5 is 16, for ASA E is 15.
UPDATE ITEM SET iResponse = 'DISCARD->#1->SET(ASA)'
    WHERE iID = 90;
UPDATE ITEM SET iResponse = 'DISCARD->#2->SET(ASA)'
    WHERE iID = 91;
UPDATE ITEM SET iResponse = 'DISCARD->#3->SET(ASA)'
    WHERE iID = 92;
UPDATE ITEM SET iResponse = 'DISCARD->#4->SET(ASA)'
    WHERE iID = 93;
UPDATE ITEM SET iResponse = 'DISCARD->#5->SET(ASA)'
    WHERE iID = 94;

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (96, 1,  'E',  'ASAE', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (123, 905, 96, 16,
          0.770, 0.450, 0.04, 0.08, 9);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (95, 3,  '0',  'AsaE', '',  1);
```

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (124, 905, 95, 16,
          0.830, 0.450, 0.100, 0.08, 0);
UPDATE ITEM SET iResponse = 'SET(ASAe)' WHERE iID = 95;
```

Now, actual patient 'admission' (Admit button code is 100).

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (100, 2,  'Admit',  'AdmBut', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (125, 905, 100, 25,
          0.720, 0.900, 0.200, 0.08, 5, 'green', 'white');
```

Here's the admission routine. If we fail, we give an alert message, otherwise we move to the INTRO menu. The value in [id] is set to the person's new ID by AdmitPatient (called by DoWholeAdmission).

```
UPDATE ITEM SET iResponse =
  '$[hospitalnumber]->QUERY(SELECT PERSDATA.pdoPerson FROM PERSDATA
   WHERE PERSDATA.pdoHospNo = ''$[]'')->
   QOK->NOT->SKIP->=FailAndReload(Duplicate patient!)->
  &DoWholeAdmission->SKIP->
  =Fail(At least enter sex, hospital number and surname!)->
   $[id]->CACHE(PROCESS.Person.$[])->
  CACHE(PROCESS:~EPOCH.Process)->MENU(INTRO)'
  WHERE iID = 100;
```

If the hospital number exists, then we use the first 3 lines to detect this and simply fail! (The code is from the response to ITEM 86).

Now let's explore the admission process ... *DoWholeAdmission*. In the following, *AdmitPatient* creates the necessary processes and epoch, returning the ID of the epoch on the stack; we then simply attach data to this last epoch. *AdmitPatient* calls *CreateAdmission* which will only work if the variable $ward is correctly populated. AdmitPatient requires the variable [id], setting the person to this value.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (109,
   '#0->$[surname]->ISNULL->NOT->SKIP->RETURN->
    $[sex]->ISNULL->NOT->SKIP->RETURN->
    $[hospitalnumber]->ISNULL->NOT->SKIP->RETURN->DISCARD->
    &AdmitPatient->
    COPY->SET(EpL)->
```

```
    copy->&KeepPersonData->
    copy->$[ASA]->#2->&RecordASA->
    $[ASAe]->#1->&RecordASA->
    &RecordWeight->
    &RecordDob->
    #0->POPMENU->DISCARD->DISCARD->
    #1',
  'DoWholeAdmission');
```

For now, we just admit. At the end we pop the current (admission menu), discarding the associated X value and menu name. This is acceptable, as on returning from DoWholeAdmission, we enter the patient data menu without further ado.

We must replace the first two lines with a validation rtn. [FIX ME!!] As already mentioned, we return zero (the initial #0) if we fail, otherwise 1. Once we've invoked *AdmitPatient*, we record observation details for the new patient. Let's look at the routines in turn.

The simple *RecordWeight* function takes the values in EpL and wt and associates them in the MEASURE table.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (252,
   '$[wt]->ISNULL->NOT->SKIP->RETURN->
    KEY(Measure)->$[EpL]->$[wt]->
    DOSQL(INSERT INTO MEASURE(measure,Epoch,meWt)VALUES($[],$[],$[]))',
            'RecordWeight');
```

If the weight variable is null, then nothing is done, otherwise the weight is recorded as an observation using the 'current epoch' value in $EpL, which must be pre-set.

Although we only use it later, let's here examine *FetchWeight*.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (253,
 'X->QUERY(SELECT MEASURE.meWt FROM MEASURE,EPOCH,PROCESS
   WHERE MEASURE.Epoch = EPOCH.epoch AND
    EPOCH.Process = PROCESS.process AND
    PROCESS.Person = $[])->QOK->SKIP->RETURN(?)->
    DIV(#1000)',
            'FetchWeight');
```

This routine assumes that X is the ID of the current patient, and obtains the weight (if present) from the MEASURE table using a join on EPOCH and PROCESS. We convert from grams to the nearest kilogram.

RecordDob is a recent function which does just that. Knowing the value in $id, we examine the $dob variable and update the relevant PERSON table entry if required:

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (266,
   '$[dob]->ISNULL->NOT->SKIP->RETURN->
    $[dob]->TIMESTAMP->$[id]->
    DOSQL(UPDATE PERSON SET pBorn=TIMESTAMP ''$[]'' WHERE person = $[])',
            'RecordDob');
```

In the above we first convert a date to a timestamp using the TIMESTAMP command. We use DOB to determine the age of a person, given X as the ID of that person:

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (267,
   'X->
    QUERY(SELECT pBorn FROM PERSON WHERE person = $[])->
    COPY->ISNULL->NOT->SKIP->RETURN->
    TIMESTAMP->FLOAT->NOW->FLOAT->SWOP->SUB->
    FLOAT(365.25)->DIV->INTEGER',
            'FetchAge');
```

The above fetches the age (returning null if the age is dud), otherwise converting the DOB sequentially to a timestamp and then a float, subtracting the current timestamp (NOW) and then dividing by 365.25 to get years.

AdmitPatient assumes that $ward is the current ward. She creates a new person ID (unique, and local to this PDA), populating the 'id' local variable with this value. In addition, the X value for the *next* menu is set to this ID.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (110, '"Person"->KEY->SET(id)->
    $[id]->setX->
    $[id]->now->
    DOSQL(INSERT INTO PERSON(person,pMade,pStatus)
    VALUES($[],TIMESTAMP ''$[]'',1))->
    &CreateAdmission',
 'AdmitPatient');
```

After inserting the person into the PERSON table,[52] we move on to create the admission process itself, for that person. Let's examine *CreateAdmission.* Here we perform three main actions:

1. Create an admission process itself (type 3),[53] with an attached epoch;

[52] A status value of 1 indicates a patient.

[53] See Table 2 in the preceding document ie. Part I.

2. Create a BED observation, using the *current* value in X, which describes the ward. We admit to a generic bed for that ward. The BED observation is attached to the observation (EPOCH) of the *admission* process, which key was returned on the stack by *ProcAndEpoch*;

3. Create an observation process (type 1, distinct from the admission one), and return the associated key of the EPOCH on the stack!

We must return the key of the *observation process* created during this routine.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (111,
    '#3->&ProcAndEpoch->BURY->
    "Badobs"->KEY->$[ward]->#10000->MUL->DIGUP->$[id]->
      COPY->
      DOSQL(UPDATE BADOBS SET boInactive=1 WHERE Person = $[])->
    DOSQL(INSERT INTO BADOBS(badobs,Bed,Epoch,Person,boFlag)
    VALUES($[],$[],$[],$[],0))->
    #1->&ProcAndEpoch',
  'CreateAdmission');
```

We bury the EPOCH, and then create a BED observation on it. We generate the generic BED from X (the current ward id) multiplied by 10000, that is times 100 for the room and 100 for the BED. As usual, we ensure that there is only one active BED for this patient within BADOBS by invoking the UPDATE BADOBS statement.

Here's the *ProcAndEpoch* routine. Given values in the local variable 'id' (for the patient) we create a new process and an observation on that process! The type of process is specified as an integer argument on the stack. The routine *returns* the ID of the created *epoch* on the stack.

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (112,
    'COPY->$[id]->DOSQL(UPDATE PROCESS SET cold=3 WHERE ProcType = $[] AND Person
    BURY->"Process"->KEY->copy->$[id]->now->now->me->DIGUP->
    DOSQL(INSERT INTO PROCESS
     (process,Person,rStart,rCreated,rPlanner,ProcType)
     VALUES($[],$[],TIMESTAMP ''$[]'',TIMESTAMP ''$[]'',$[],$[]))->
    "Epoch"->KEY->copy->BURY->SWOP->now->me->
    DOSQL(INSERT INTO EPOCH(epoch,Process,oMade,Person)
    VALUES($[],$[],TIMESTAMP ''$[]'',$[]))->DIGUP',
   'ProcAndEpoch');
```

Here we record personal (admission) data. We submit the process epoch on the stack:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (113,
 'BURY->
 "Persdata"->KEY->DIGUP->
  $[surname]->$[forename]->$[hospitalnumber]->
  $[sex]->$[id]->
 DOSQL(INSERT INTO PERSDATA(persdata,Epoch,
   pdoSurname,pdoForename,pdoHospNo,
   pdoGender,pdoPerson)
  VALUES($[],$[],''$[]'',''$[]'',''$[]'',$[],$[]))',
   'KeepPersonData');
```

We use our knowledge that $[id] contains the local database ID number of the person. Here's the ASA rating function. On the top of the stack, we submit an integer 2 if we're recording the ASA 1–5, or a 1 if we're recording the 'E' rating:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (114,
   '#3->MARK->SWOP->URZN->SWOP->
   "Medscore"->KEY->
   DOSQL(INSERT INTO MEDSCORE(Epoch,msoValue,msoNature,medscore)
   VALUES($[],$[],$[],$[]))->
   UNMARK',
 'RecordASA');
```

The first line marks the stack, and then quits the routine without doing anything if the ASA value is NULL (using URZN to test for this case). We have a similar problem to that in KeepPersonData. The fix:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (114,
   '#3->MARK->SWOP->URZN->SWOP->
   BURY->BURY->BURY->
   "Medscore"->KEY->
   DIGUP->DIGUP->DIGUP->
   DOSQL(INSERT INTO MEDSCORE(medscore,Epoch,msoValue,msoNature)
   VALUES($[],$[],$[],$[]))->
   UNMARK',
 'RecordASA');
```

5.4 Finding a patient (919)

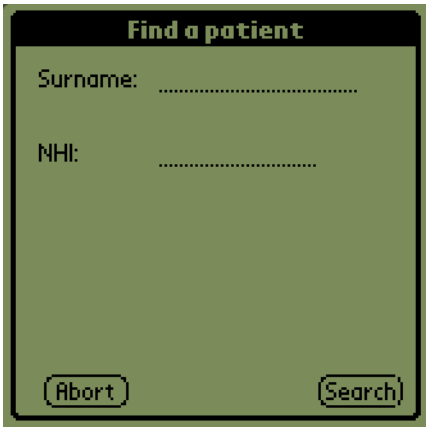


Figure 4: Find a patient

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (919, 20, 'Identify patient(s)', 'SEARCH');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (919, 919, 919, 0,
          0.001, 0.001, 0.990, 0.990, 0);

UPDATE ITEM SET iInitial =
  'NAME(Surname)->NAME(NHI)->NAME(id)->
  NAME(ward)->#1->SET(ward)'
  WHERE iID = 919;
```

Because we may not necessarily know the ward even when we identify the patient (as the patient may not have a ward, having been discharged) we set a default ‘unknown’ ward of one.

We use the ‘Exit’ button (similar to ‘Abort’ code 80). We create our own Surname and Hospital Number fields.

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1210, 2,  'Exit',  'x', '',  1);
UPDATE ITEM SET iResponse = 'MENU(1)' WHERE iID = 1210;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1210, 919, 1210, 1,
          0.070, 0.900, 0.200, 0.08, 6, 'red', 'white');

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
```

```
          VALUES (1200, 1,  'Surname: ',  'asur', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1200, 919, 1200, 2,
          0.050, 0.62, 0.200, 0.08, 0);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1201, 10,  '',  'SurTxt', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1201, 919, 1201, 3,
          0.350, 0.62, 0.500, 0.08, 0);
UPDATE ITEM SET iResponse = 'SET(Surname)' WHERE iID = 1201;

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1202, 1,  'NHI: ',  'HNo', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1202, 919, 1202, 6,
          0.050, 0.750, 0.100, 0.08, 0);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1203, 10,  '',  'TxNhi', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1203, 919, 1203, 7,
          0.350, 0.750, 0.400, 0.08, 0);
UPDATE ITEM SET iResponse = 'UPPERCASE->SET(NHI)'
    WHERE iID = 1203;
--- later should validate this

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1205, 2,  'Search',  'srch', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1205, 919, 1205, 10,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
```

Here's the script for the search button.

```
UPDATE ITEM SET iResponse = '$[NHI]->
  COPY->ISNULL->NOT->SKIP->=GoSurname->
  QUERY(SELECT PERSDATA.pdoPerson FROM PERSDATA
    WHERE PERSDATA.pdoHospNo = ''$[]'')->QOK->
    SKIP->=Fail(Not found)->copy->SETX->
    CACHE(PROCESS.Person.$[])->
  CACHE(PROCESS:~EPOCH.Process)->MENU(INTRO)'
  WHERE iID = 1205;
```

Here's *GoSurname* which, if the surname is filled in, tries to find matching surnames. We must create the separate SURNAME menu (next section) to which we pass the surname as X, and then in this new menu we will search, creating a list of candidates similar to a ward list!

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (235,
  '$[Surname]->
  COPY->ISNULL->NOT->SKIP->=Fail(Please enter Surname or NHI)->
  QUERY(SELECT persdata FROM PERSDATA
    WHERE pdoSurname = ''$[]'')->QOK->
    SKIP->=Fail(No surname match!)->
    SETX->MENU(SURNAME)',
 'GoSurname');
```

The above tests for *at least one* patient with this identical name, otherwise failing miserably! See how, in keeping with our current convention that X can only be a number, we return the PERSDATA id of just *one* such name.

5.4.1 Finding patients who haven't been seen

A simple button invokes this function:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1206, 2,  'Flag unseen!',  'fp', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1206, 919, 1206, 10,
          0.30, 0.03, 0.400, 0.08, 0);

UPDATE ITEM SET iResponse = '&FlagRecent->Alert(Unseen flagged -!-)->MENU(1)'
  WHERE iID = 1206;
```

5.4.2 Finding patients with recent 'problems'

Another button:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1207, 2,  'Flag problems!',  'fp', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1207, 919, 1207, 10,
          0.30, 0.16, 0.400, 0.08, 0);

UPDATE ITEM SET iResponse = '&FlagProblem->Alert(Problems flagged -!-)->MENU(1)'
  WHERE iID = 1207;
```

5.4.3 Find those marked for ‘PM review’

Again, almost identical:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1208, 2,  'For PM review!',  'fp', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1208, 919, 1208, 10,
          0.30, 0.29, 0.400, 0.08, 0);

UPDATE ITEM SET iResponse = '&FlagPM->Alert(PM Flagged -!-)->MENU(1)'
  WHERE iID = 1208;
```

We also need a reminder to ‘Exit’ and use the usual menu system:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1209, 1,  '(Exits to main menu once flagged)',  'rem', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1209, 919, 1209, 6,
          0.050, 0.42, 0.900, 0.08, 0);
```

5.5 Selection from a list of surnames (918)

Figure 5: Selection from a list

Here’s the first draft of *SearchBySurname*:

We must create a separate menu to which we pass the PERSDATA surname id as X, and then in this new menu we will search, creating a list of candidates similar to a ward list!

It would be simple, quick and fun to search thus . . .

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (236,
 'X->QUERY(SELECT pdoSurname FROM PERSDATA
  WHERE persdata = $[])->
  QMANY(SELECT PERSDATA.pdoPerson FROM PERSDATA
    WHERE PERSDATA.pdoSurname = ''$[]'')',
 'SearchBySurname');
```

. . . buuut we have a teensy problem: we will also include staff members in our search! We therefore use the cumbersome join:

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (236,
 'X->QUERY(SELECT pdoSurname FROM PERSDATA
  WHERE persdata = $[])->
  QMANY(SELECT PERSDATA.pdoPerson FROM PERSDATA,EPOCH,PROCESS,PERSON
    WHERE PERSDATA.pdoSurname = ''$[]'' AND
    PERSDATA.Epoch = EPOCH.epoch AND
    EPOCH.Process = PROCESS.process AND
    PROCESS.Person = PERSON.person AND
    PERSON.pStatus < 2)',
 'SearchBySurname');
```

The search is still quick because the second SELECT balks before the first AND if the surnames don't match; if there is a match we track the joins to see whether the person is a patient or not!

This menu is *very* similar to the selection of a patient within a ward. Button 80 is the Abort button, and 1230 is a polymorphic table describing the patients.

```
INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (918, 20, 'Select on Surname', 'SURNAME');

INSERT INTO ITEM (iID, iType, iText, iName, iLines)
  VALUES (1230,8,'[Not found]','PtTbl',10);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (1220, 918, 918, 0, 0.001, 0.001, 0.990, 0.990),
         (1221, 918, 1230,  1, 0.001, 0.001, 0.999, 0.850),
         (1223, 918, 80,  3, 0.200, 0.900, 0.200, 0.080);

INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (1226, 1,  'w',  'Wd');

INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (1227,  2,     'i',    'Id' );
```

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1228,  1,     's',    'Snx' );

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
                       irName, irFraction, irEnabled)
  VALUES (1226, 1230, 1226, 3, 'Ward',      0.25, 1),
         (1227, 1230, 1227, 2, 'ID',      0.30, 1),
         (1228, 1230, 1228, 1, 'Surname', 0.45, 1);

UPDATE ITEM SET iInitial  = 'V->&GetPatientWard'
  WHERE iID =1226;

UPDATE ITEM SET iInitial = '&SearchBySurname'
  WHERE iID = 1230;

UPDATE ITEM SET iInitial  = 'V->&FetchIdNumber'
  WHERE iID =1227;

UPDATE ITEM SET iInitial  = 'V->&FetchSurname'
  WHERE iID = 1228;

UPDATE ITEM SET iResponse =  'V->COPY->SETX->CACHE(PROCESS.Person.$[])->
  CACHE(PROCESS:~EPOCH.Process)->MENU(INTRO)'
  WHERE iID = 1227;
```

In setting up the SURNAME menu, we don't know whether the patient will have a ward, so we play it safe and create a ward of 1 (unknown)!

```
UPDATE ITEM SET iInitial =
  'NAME(id)->
   NAME(ward)->#1->SET(ward)'
  WHERE iID = 918;
```

The response button for the ID button (1227) is similar to that for item 21, invoking *EnterDetailMenu*. V is the patient ID.

Here's the *GetPatientWard* function, very similar to *GetPatientRoom*:

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(237,

'QUERY(SELECT BADOBS.Bed FROM BADOBS
    WHERE BADOBS.Person = $[] AND BADOBS.boInactive IS NULL)->
   QOK->SKIP->RETURN(?)->
  #10000->DIV->INTEGER->
QUERY(SELECT WARD.swrdText FROM WARD
 WHERE WARD.ward = $[])',

'GetPatientWard');
```

5.6 Patient alert screen (903)

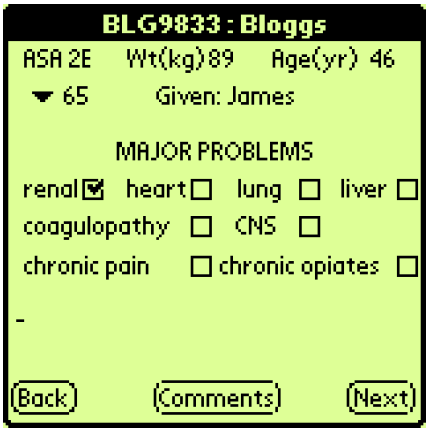


Figure 6: Patient alerts

This menu, with the arbitrary code 903, has an X (transfer variable) value set to the unique ID of the current patient.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (903, 20, 'Alerts', 'DETAILS');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (4, 903, 903, 0,
            0.001, 0.001, 0.990, 0.990, 0);
```

Here's the initialisation of the DETAILS menu (903):

```
UPDATE ITEM SET iInitial =
 'X->&FetchIdNumber->X->&FetchSurname->
  X->&GetBed->
 "$[] : $[] ($[])"->Title->
   NAME(EpL)->#1->X->&LastEpoch->SET(EpL)'
 WHERE iID = 903;
```

Here's GetBed:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (297,
   'QUERY(SELECT Bed FROM BADOBS WHERE Person = $[] AND boInactive IS NULL)->DIV(#
  COPY->DIV(#100)->MUL(#100)->SUB->COPY->SAME(#0)->NOT->SKIP->REPLACE('?')',
        'GetBed');
```

Continuing ...

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (132, 2,  'Not seen',  'ns', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (32, 903, 132, 1,
          0.003, 0.910, 0.230, 0.08, 0);
-- back button

UPDATE ITEM SET iResponse =
   'CONFIRM(Are you sure? [Data may be lost])->
     SKIP->RETURN->
     MENU(-1)' WHERE iID = 132;
```

In the above we abort the edit and return to the previous menu, also turning off caching. We formerly invoked EndEpoch, but this was wrong, as we should *not* record the interval if the patient was 'not seen' as flagging will then fail!

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (133, 2,  'Next',  'Continuation button', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
       miX, miY, miW, miH, miGroup, miPaper)
    VALUES (33, 903, 133, 2,
       0.820, 0.910, 0.160, 0.08, 0, 'green');
```

Here's the 'Next button' response.

```
UPDATE ITEM SET iResponse =
  'X->&Is24->BOOLEAN->SKIP->MENU(PAINDATA)->MENU(RGNCHECK)'
  WHERE iID = 133;
```

We must look carefully at the above. If we access the PAINDATA menu from elsewhere, we should similarly invoke RGNCHECK if Is24 returns true. At present this is the only access point.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (140, 1,  'renal',  're');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (40, 903, 140, 3,
          0.03, 0.080, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (144, 1,  'coagulopathy',  'co');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (44, 903, 144, 3,
          0.03, 0.160, 0.20, 0.08, 0);
```

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (142, 1,  'chronic pain',  'cp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (42, 903, 142, 3,
          0.03, 0.240, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (143, 1,  'lung',  'lu');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (43, 903, 143, 3,
          0.55, 0.080, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (145, 1,  'chronic opiates',  'ch');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (45, 903, 145, 3,
          0.50, 0.240, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (146, 1,  'heart',  'hr');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (46, 903, 146, 3,
          0.28, 0.080, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (147, 1,  'liver',  'li');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (47, 903, 147, 3,
          0.80, 0.080, 0.20, 0.08, 0);
```

On request, we've also added the patient weight in over here; we also add a 'CNS dysfunction' tickbox.

In the following section we specify the ini scripts and responses for the various tickboxes. The numbers specify the process IDs for the various problems, for example #1050 is 'coagulopathy'.[54]

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (149, 1,  'MAJOR DYSFUNCTION:',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
```

[54] Consult the relevant tables in *AnalgesiaDBpart1*.

```
          miX, miY, miW, miH, miGroup)
  VALUES (49, 903, 149, 3,
          0.03, 0.01, 0.50, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (150, 3,  '',  're');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (50, 903, 150, 4,
          0.16, 0.080, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1000->&WasItOn' WHERE iID = 150;
UPDATE ITEM SET iResponse = '#1000->SWOP->&OnOrOff' WHERE iID = 150;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (156, 3,  '',  'hr');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (56, 903, 156, 6,
          0.42, 0.080, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1010->&WasItOn' WHERE iID = 156;
UPDATE ITEM SET iResponse = '#1010->SWOP->&OnOrOff' WHERE iID = 156;


INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (153, 3,  '',  'lu');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (53, 903, 153, 8,
          0.68, 0.080, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1020->&WasItOn' WHERE iID = 153;
UPDATE ITEM SET iResponse = '#1020->SWOP->&OnOrOff' WHERE iID = 153;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (157, 3,  '',  'li');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (57, 903, 157, 10,
          0.92, 0.080, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1030->&WasItOn' WHERE iID = 157;
UPDATE ITEM SET iResponse = '#1030->SWOP->&OnOrOff' WHERE iID = 157;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (154, 3,  '',  'co');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (51, 903, 154, 12,
          0.42, 0.160, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1050->&WasItOn' WHERE iID = 154;
UPDATE ITEM SET iResponse = '#1050->SWOP->&OnOrOff' WHERE iID = 154;
```

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (166, 1,  'CNS',  'cn');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (166, 903, 166, 14,
          0.54, 0.160, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (165, 3,  '',  'cns');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (165, 903, 165, 16,
          0.68, 0.160, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1090->&WasItOn' WHERE iID = 165;
UPDATE ITEM SET iResponse = '#1090->SWOP->&OnOrOff' WHERE iID = 165;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (152, 3,  '',  'cp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (52, 903, 152, 18,
          0.42, 0.240, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1060->&WasItOn' WHERE iID = 152;
UPDATE ITEM SET iResponse = '#1060->SWOP->&OnOrOff' WHERE iID = 152;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (155, 3,  '',  'opi');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (55, 903, 155, 20,
          0.92, 0.240, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1070->&WasItOn' WHERE iID = 155;
UPDATE ITEM SET iResponse = '#1070->SWOP->&OnOrOff' WHERE iID = 155;
```

In July 2008 we added diabetes and 'neoplasia' to our list, and adjusted the positions of the items below, in consequence:

```
INSERT INTO PROCTYPE (proctype, rptNature)
  VALUES ( 1075, 'diabetes' ),
    ( 1085, 'neoplasm' );

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (148, 1,  'diabetes',  'd');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (148, 903, 148, 3,
```

```
          0.03, 0.320, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (167, 3,  '',  'd');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (167, 903, 167, 20,
          0.42, 0.320, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1075->&WasItOn' WHERE iID = 167;
UPDATE ITEM SET iResponse = '#1075->SWOP->&OnOrOff' WHERE iID = 167;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (151, 1,  'neoplasm',  'n');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (151, 903, 151, 3,
          0.50, 0.320, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (168, 3,  '',  'n');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (168, 903, 168, 20,
          0.92, 0.320, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '#1075->&WasItOn' WHERE iID = 168;
UPDATE ITEM SET iResponse = '#1075->SWOP->&OnOrOff' WHERE iID = 168;

-- move SURGERY: text down a shade, as well as the associated menu (651):

UPDATE MENUITEMS SET miY = 0.40 WHERE miUid = 1149;

UPDATE MENUITEMS SET miY = 0.47 WHERE miUid = 651;
```

We need to insert a list of operations:

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1149, 1,  'SURGERY:',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1149, 903, 1149, 3,
          0.03, 0.38, 0.36, 0.08, 0);


INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (650, 8,  '[No operation]',  'Cmnt', '',  5);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
```

```
          VALUES (622, 1,  '-',  'Typ1', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (621, 1,  '-',  'Dat1', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (624, 1,  '-',  'c1', '',  1);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
   miX, miY, miW, miH, miGroup, miEnabled)
  VALUES (651, 903, 650, 22,
   0.001, 0.450, 0.999, 0.400, 1, 0);
-- group of 1 = 'clickable'

--INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
--            irName, irFraction)
--    VALUES (654, 650, 623, 3,
--            'Op. Site', 0.40);
--- site of surgery (disabled)

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
            irName, irFraction)
    VALUES (652, 650, 621, 1,
            'Date', 0.24);
--- date of observation

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
            irName, irFraction)
    VALUES (653, 650, 622, 2,
            'Op. Type', 0.34);
--- type of surgery

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
            irName, irFraction)
    VALUES (655, 650, 624, 3,
            'Note', 0.42);
--- date of observation
```

Here's the response to clicking on an operation name:

```
UPDATE ITEM SET iResponse =
  'V->QUERY(SELECT COMMENT.cText FROM COMMENT
    WHERE COMMENT.Epoch = $[])->Alert'
  WHERE iID =624;

UPDATE ITEM SET iInitial = 'X->QMANY(SELECT EPOCH.epoch
    FROM EPOCH,PROCESS
    WHERE EPOCH.Process = PROCESS.process AND
    PROCESS.ProcType = 500 AND
    PROCESS.Person = $[])'
  WHERE iID = 650;
```

Finally we have the button to ‘add operations’:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (245, 2,  'Add operation',  'AdCmt', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (245, 903, 245, 40,
          0.52, 0.41, 0.44, 0.08, 0);
UPDATE ITEM SET iResponse = 'MENU(SURGERY)' WHERE iID = 245;
-- Add operation button
```

5.6.1 LastEpoch

LastEpoch identifies the most recent epoch on the stated type of process for this patient. The top item on the stack is the patient, and deep to this is the type of process, e.g. 1 for a general observation process. The reasonable assumption is made that this process has already been created (See *EnterDetailMenu*).

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (163,
    'QMANY(SELECT MAX(EPOCH.epoch) FROM EPOCH,PROCESS
     WHERE EPOCH.Process = PROCESS.process AND
     PROCESS.ProcType = $[] AND
     PROCESS.Person = $[] AND
     PROCESS.rEnd IS NULL)->QOK->SKIP->#0->RETURN',

        'LastEpoch');
```

5.6.2 Patient ward: get and set

We must be able to obtain and change the patient’s current ward from within the ‘Patient Alerts’ menu. The routines are respectively *GetMyWard* and *SetNewWard*.

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(169,
 'X->QUERY(SELECT BADOBS.Bed
    FROM BADOBS WHERE BADOBS.Person = $[] AND BADOBS.boInactive IS NULL)->
  #10000->DIV->INTEGER->
  COPY->SET(ward)->
  QUERY(SELECT WARD.swrdText FROM WARD
    WHERE WARD.ward = $[])',

'GetMyWard');
```

We have modified *GetMyWard* to store the ward ID in the [ward] variable, a bit of a hack! This variable must now exist. The second INTEGER command is

a consequence of Perl's weak typing: on the desktop (without the command) we start looking for a floating point value!

The next section sets a new ward (with an unknown BED) for the current patient. Already on the stack is the ID of the ward and the patient ('Person' ID). We first (after transiently burying the patient ID) confirm the move, and reload the menu on failure. Note that reloading the menu will terminate the script, and clear the buried datum. On success, we dig up the patient and continue.

Then, deepest of all, we bury a copy of the person ID. We then bury the ward ID after multiplying by 10000 to convert the ward ID to a generic BED ID for that ward! Using the ID of the patient we find the process with a type of 3 (admission process), create a new epoch for that process, and then make a 'badobs' to document a 'generic BED' for that ward.

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(170,
 'BURY->
  COPY->QUERY(SELECT swrdText FROM WARD WHERE ward = $[])->
  CONFIRM(Move to Wd $[]?)->SKIP->MENU(#0)->
 DIGUP->
 COPY->BURY->
 SWOP->#10000->MUL->BURY->
   QUERY(SELECT PROCESS.process
     FROM PROCESS WHERE PROCESS.Person = $[]
     AND PROCESS.ProcType = 3)->BURY->
   KEY(Epoch)->COPY->NOW->ME->DIGUP->
   DOSQL(INSERT INTO EPOCH(epoch,oMade,Person,Process)
     VALUES($[],TIMESTAMP ''$[]'',$[],$[]))->
 BURY->
 KEY(Badobs)->DIGUP->DIGUP->DIGUP->
      COPY->
      DOSQL(UPDATE BADOBS SET boInactive=1 WHERE Person = $[])->
   DOSQL(INSERT INTO BADOBS(badobs,Epoch,Bed,Person,boFlag)
     VALUES($[],$[],$[],$[],1))',
'SetNewWard');
```

We set boFlag to 1 by default, as after moving a patient around, we regard them as 'interesting'! See the two invocations of *SetNewWard*.

5.6.3 Obtaining patient data

We've previously defined our *FetchSurname* routine and we use it to good effect over here. Very similar is *FetchForename*:

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(154,

'QMANY(SELECT MAX(PERSDATA.persdata) FROM PERSDATA
```

```
  WHERE PERSDATA.pdoPerson = $[])->
QUERY(SELECT PERSDATA.pdoForename FROM PERSDATA
  WHERE PERSDATA.persdata = $[])',

'FetchForename');
```

Similar in some respects is fetching ASA data. Assuming X is the current patient, we submit the 'medical code' (here 1 or 2 for ASA/ASAe), and laboriously retrieve the correct value. First the *FetchMedscore* routine:

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(155,

'X->
 QMANY(SELECT MAX(MEDSCORE.medscore) FROM MEDSCORE,EPOCH,PROCESS
  WHERE MEDSCORE.Epoch = EPOCH.epoch
   AND MEDSCORE.msoNature = $[]
   AND EPOCH.Process = PROCESS.process
   AND PROCESS.Person = $[])->QOK->SKIP->RETURN(?)->
 QUERY(SELECT MEDSCORE.msoValue FROM MEDSCORE
  WHERE MEDSCORE.medscore = $[])',

'FetchMedscore');
```

...and next the actual ASA retrieval:

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(156,

'#2->&FetchMedscore->
 #1->&FetchMedscore->
 #1->SAME->NOT->SKIP->"$[]E"->RETURN',

'FetchASA');
```

First we fetch the ASA value in the range of 1–5, then the E score, coded as a 0 or a 1. If the E value is 1 (i.e. 'yes') then only do we add an 'E' suffix.

5.6.4 Finding the most recent process

The routine *FindRecentProcess* is a useful, generic routine which takes the current patient (as X) and, given a process code on the stack, returns the ID of the most recent such process for that patient. (Of course the query fails and returns zero if no such process exists).

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (115, 'X->
    QMANY(SELECT MAX(PROCESS.process) FROM PROCESS WHERE
       PROCESS.ProcType = $[] AND PROCESS.Person = $[]
       AND PROCESS.rEnd IS NULL)',
 'FindRecentProcess');
```

The process code (ProcType) is assumed to be on the stack, and X is placed there. If the SELECT statement failed, we return zero, otherwise the result of the query.

5.6.5 Checking active processes

The *WasItOn* routine is similar to the above. Given the same information, we determine the rEnd value, returning 1 if it's null and otherwise returning zero!

Here we use this routine to find whether there are major problems such as renal failure, as each process is modelled as a problem. If the process has terminated, the termination timestamp is filled in, but otherwise the timestamp is null.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (116, 'X->
    QMANY(SELECT PROCESS.process FROM PROCESS WHERE
       PROCESS.rEnd IS NULL AND
       PROCESS.ProcType = $[] AND PROCESS.Person = $[])->
       QOK->SKIP->RETURN(#0)->
       #1',
 'WasItOn');
```

If no process exists, return zero. If the process exists and the timestamp is null, then only return 1.

Similar is the *WasBetween* routine, which identifies any single epoch which is both *more recent* than EpL (the 'most recent general epoch'), and on an *active* process between the two submitted codes! The first ProcType code submitted must be smaller than the second.

The idea is that in the current epoch (which starts with EpL) we will have observations on (active) processes only if the relevant menu has been visited.

How do we document, for example, that we've noted that *no* epidural is in? We don't want to create a virtual (non-epidural) process so this seems to fall logically into the field of a general observation! Note that there can be conflict between such an observation and an ongoing epidural process — this needs to be resolved!

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (173,
    'BURY->BURY->$[EpL]->X->DIGUP->DIGUP->
     QUERY(SELECT EPOCH.epoch FROM EPOCH,PROCESS WHERE
       EPOCH.epoch > $[] AND
       EPOCH.Process = PROCESS.process AND
       PROCESS.rEnd IS NULL AND
       PROCESS.Person = $[] AND
       PROCESS.ProcType > $[] AND
       PROCESS.ProcType < $[])->
```

```
     QOK->SKIP->RETURN(#0)->
     DISCARD->#-1',
 'WasBetween');
```

5.6.6 Ending a process

Given a process TYPE (ProcType), and the patient in X, terminate the process. The following assumes that there is only one such active process (and will only terminate the first such process it finds).

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (117, 'X->
    QUERY(SELECT PROCESS.process FROM PROCESS WHERE
       PROCESS.ProcType = $[] AND PROCESS.Person = $[]
       AND PROCESS.rEnd IS NULL)->
       QOK->SKIP->RETURN->BURY->NOW->DIGUP->
    DOSQL(UPDATE PROCESS SET rEnd=TIMESTAMP ''$[]'' WHERE process = $[])',
 'EndProcByType');
```

5.6.7 Creating a new process

NewProc is convenience function which creates a new process. We *assume* that X contains the ID of the patient, and that the type of process is on the stack. The process ID is returned on the stack.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (118,
    'BURY->KEY(Process)->copy->X->now->now->me->DIGUP->
    DOSQL(INSERT INTO PROCESS
     (process,Person,rStart,rCreated,rPlanner,ProcType)
     VALUES($[],$[],TIMESTAMP ''$[]'',TIMESTAMP ''$[]'',$[],$[]))',

   'NewProc');
```

5.6.8 Dated Procedure

DatedProc resembles *NewProc* but we provide a timestamp for the rStart database value on the stack above the procedure type! The stack items which must be submitted to this ugly procedure are, in order from deep to most superficial are:

- patient
- process type
- timestamp

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (185,
    'BURY->BURY->BURY->
    KEY(Process)->NOW->ME->DIGUP->DIGUP->DIGUP->
    DOSQL(INSERT INTO PROCESS
     (process,rCreated,rPlanner,Person,ProcType,rStart)
     VALUES($[],TIMESTAMP ''$[]'',$[],$[],$[],TIMESTAMP ''$[]''))',

   'DatedProc');
```

5.6.9 On or off?

This simple routine, *OnOrOff* accepts a one or zero on the top of the stack, with a process code below. If the value is (now) on, then a new process is created, but if it's off, the current process is closed!

[fix me using the =FXNAME convention ?????????????]

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (119, 'COPY->BURY->SKIP->&EndProcByType->
      DIGUP->SKIP->RETURN->&NewProc->DISCARD',
 'OnOrOff');
```

5.7 General Comments (906)

Figure 7: General comments

We attach one or more comments to the most recent epoch on a type 1 Process (A type 1 process records 'general' observations). These comments are used as general alerts, communicating important information between PDA users. If there is no such epoch, one is created.

Let's look at the *comments subtable* which lists all comments for a particular patient, and the date on which they were made:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
         VALUES (643, 8,  '[No comments]',  'Cmnt', '',  10);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
         VALUES (644, 1,  '-',  'Date1', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
         VALUES (645, 1,  '-',  'Note2', '',  1);

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (14, 643, 644, 1, 'Date', 0.25);
--- date of epoch column (we discard time component of timestamp)

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (15, 643, 645, 2, 'Comment', 0.75);
--- observation comment (free text)
```

Here's the response to clicking on a comment line — it allows us to display the whole datum as an Alert.

```
UPDATE ITEM SET iResponse =
       'V->QUERY(SELECT COMMENT.cText FROM COMMENT
           WHERE COMMENT.comment = $[])->Alert'
       WHERE iID =645;
```

Here's the similar timestamp response:

```
UPDATE ITEM SET iResponse =
       'V->QUERY(SELECT EPOCH.oMade FROM COMMENT,EPOCH
           WHERE COMMENT.comment = $[] AND COMMENT.Epoch = EPOCH.epoch)->Alert'
       WHERE iID =644;

UPDATE ITEM SET iInitial =
       'X->QMANY(SELECT COMMENT.comment FROM COMMENT,EPOCH,PROCESS
          WHERE COMMENT.Epoch = EPOCH.epoch AND EPOCH.Process = PROCESS.process
          AND PROCESS.Person = $[] ORDER BY COMMENT.comment DESC)'
       WHERE iID = 643;

UPDATE ITEM SET iInitial =
       'V->QUERY(SELECT EPOCH.oMade FROM COMMENT,EPOCH
           WHERE COMMENT.comment = $[] AND COMMENT.Epoch = EPOCH.epoch)->
           SPLIT( )->DISCARD->&ShortDate'
       WHERE iID =644;

UPDATE ITEM SET iInitial =
       'V->QUERY(SELECT COMMENT.cText FROM COMMENT
           WHERE COMMENT.comment = $[])'
       WHERE iID =645;
```

Next, we have the actual comment *menu*, which includes the comments table (643) as previously specified.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (906, 20, 'Comment', 'COMMENTS');
--- set busyBottom, as well as todo flag. X is patient uid.

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (660, 906, 906, 0,
            0.001, 0.001, 0.990, 0.990, 0);
--- Self-reference.

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup, miEnabled)
  VALUES (661, 906, 643, 22,
            0.001, 0.140, 0.999, 0.760, 1, 0);
--- above is comments table(643)!
-- group of 1 forces all comments to be clickable!!

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (660, 2,  'Done',  'done', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (662, 906, 660, 10,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');

-- also have a 'more' button: [2007-11-01]
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (665, 906, 9926, 99,
          0.02, 0.900, 0.16, 0.08, 0, 'yellow', 'black');
```

Here's the response to clicking on the [Done] button. If the comment hasn't yet been stored, we provide a friendly reminder; otherwise we leave the menu.

```
UPDATE ITEM SET iResponse =
  '$[comment]->ISNULL->NOT->SKIP->MENU(-1)->
  CONFIRM(Abandon comment?)->NOT->SKIP->MENU(-1)'
  WHERE iID = 660;
```

Continuing …

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (661, 10,  '',  'ctxt', '',  1);
UPDATE ITEM SET iResponse = 'set(comment)' WHERE iID = 661;
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
```

```
          VALUES (662, 2,  'Add',  'cbtn', '',  1);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (663, 906, 661, 3,
          0.001, 0.050, 0.790, 0.08, 0);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (664, 906, 662, 4,
          0.820, 0.050, 0.170, 0.10, 0);
```

Here's the initialisation:

```
UPDATE ITEM SET iInitial =
 'name(comment)->
 X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
  WHERE iID = 906;
```

And a response to clicking [Add]:

```
UPDATE ITEM SET iResponse =
  '$[comment]->ISNULL->NOT->SKIP->
    =Fail(Please fill in comment field first!)->
    &MakeGeneralComment->MENU(#0)'
  WHERE iID = 662;
```

If *MakeGeneralComment* succeeds, we reload the menu to update the comment display!

We also need a check (with several other buttons) to ensure that we don't abandon a completed comment if we've forgotten to press [Add] and move elsewhere:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (291,
    '$[comment]->ISNULL->NOT->SKIP->RETURN->
    CONFIRM(Abandon comment?)->SKIP->&MakeGeneralComment->RETURN',
        'OkComment');
```

Finally a function ...

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (150,
        '#0->MARK->
        #1->&FindRecentProcess->URZN->
        &ForceEpoch->KEY(Comment)->SWOP->
        $[comment]->&FixSQL->
        DOSQL(INSERT INTO COMMENT(comment,Epoch,cText)
              VALUES($[],$[],''$[]''))',
        'MakeGeneralComment');
```

Here's FixSQL which duplicates the single quotes. Because we are inserting SQL into SQL, the SPLIT and JOIN commands have double the number of single quotes required! There is a (rather theoretical) potential issue with the MARK not being balanced by an UNMARK, but this should be ok as we terminate in a MENU command which clears the stack. Note the use of the deprecated (but convenient) URZN.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (271,
   'MARK(#1)->SPLIT('')->JOIN('''')->BURY->UNMARK->DIGUP',
   'FixSQL');
```

We haven't yet defined *ForceEpoch*, which accepts the ID of a process, and finds the most recent epoch on that process. If the epoch doesn't exist, then a new epoch is created! Here goes ...

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (151,
        'COPY->QMANY(SELECT MAX(EPOCH.epoch) FROM EPOCH
            WHERE EPOCH.Process = $[])->
        QOK->SKIP->&FancyEpoch->SWOP->DISCARD',

        'ForceEpoch');
```

In the above, *FancyEpoch* is only invoked if the SQL failed to place anything on the stack, in which case this function remedies the error.

FancyEpoch accepts the ID of the process to which the new epoch is to be attached. This routine is unusual in that it *leaves* the process ID on the stack, and adds a new epoch ID on top of this.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (152,
        'COPY->BURY->
        KEY(Epoch)->COPY->NOW->ME->DIGUP->
        DOSQL(INSERT INTO EPOCH(epoch,oMade,Person,Process)
         VALUES($[],TIMESTAMP ''$[]'',$[],$[]))',

        'FancyEpoch');
```

We almost forgot the *ShortDate* routine. Here we take a date in the format 'YYYY-MM-DD' and convert it to 'DD.MM.YY', for purposes of concise display only. We use the 'days first' short format as per local (New Zealand!) conventions; this routine is easily modified for other regions.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (157,
        'copy->isnull->not->skip->return->
        "-"->SPLIT->BURY->BURY->INTEGER->#2000->
        SUB->
        DIGUP->INTEGER->DIGUP->INTEGER->
        SWOP->"$[].$[]"->SWOP->"$[].0$[]"',

        'ShortDate');
```

Here's *TinyStamp* which is even more brief, omitting the year! We convert YYYY-MM-DD HH:MM:SS to dd/mm. If NULL is submitted, we return NULL.

```
 INSERT INTO FUN (fKey, fBody, fName)
   VALUES (289,
    'copy->isnull->not->skip->return->
    SPLIT( )->DISCARD->
    "-"->SPLIT->BURY->BURY->DISCARD->
    DIGUP->INTEGER->DIGUP->INTEGER->
    SWOP->"$[]/$[]"',

  'TinyStamp');
```

5.8 Pain data (907)

Figure 8: Pain data menu

This menu is the meat of our whole enterprise. We enquire about pain and its associations — these include the ability to cough effectively, the type of analgesia,

and the nature of the surgery provoking the pain (where appropriate). We start with the menu, and its forward and back ('Done') buttons.[55]

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (907, 20, 'Pain Data', 'PAINDATA');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (200, 907, 907, 0,
            0.001, 0.001, 0.990, 0.990, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (201, 2,  'Back',  'Back button', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (201, 907, 201, 39,
          0.003, 0.900, 0.150, 0.08, 0);
UPDATE ITEM SET iResponse = 'MENU(1)' WHERE iID = 201;
-- back button

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (202, 2,  'Done',  'Contin', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup, miPaper)
    VALUES (202, 907, 202, 41,
            0.820, 0.900, 0.160, 0.08, 0, 'green');
UPDATE ITEM SET iResponse = '&OkComment->Menu(FINISH)'  WHERE iID = 202;
-- 'Next' button
```

5.8.1 A new check

As of version 0.95, we amend the 'Done' button fairly considerably. Based on field testing, users tend to skip over some menus once the modality has stopped. Although we're reluctant to indulge in coercion, the following helps — we refuse to continue if a modality is currently active *and* the relevant menu hasn't been visited by clicking on the relevant [Y] button:

```
UPDATE ITEM SET iResponse =
'X->#99->#159->&IsProc->#1->&NoEvent->
   AND->NOT->SKIP->=Fail(Please visit regional menu)->
 X->#389->#391->&IsProc->#2->&NoEvent->
   AND->NOT->SKIP->=Fail(Please visit PCA menu)->
 X->&IsKetamine->#4->&NoEvent->
   AND->NOT->SKIP->=Fail(Please visit "Other" [ketamine] menu)->
 &OkComment->Menu(FINISH)' WHERE iID = 202;
```

[55] Initially we specified the *consultant present*, but we have altered the screen to remove this item.

The range of 99–159 (exclusive) looks for regional processes, while 390 is IV PCA. The corresponding event codes are 1 and 2 (to show entry into the relevant menus); ketamine is slightly different, and is present in the 'other' menu associated with event #4. Here are the three associated functions:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (294,
    'BURY->$[EpL]->DIGUP->
     QUERY(SELECT NONEVENT.noValue FROM NONEVENT WHERE
       NONEVENT.Epoch = $[] AND
       NONEVENT.noCode = $[])->
     QOK->NOT->COPY->BURY->SKIP->DISCARD->DIGUP',
    'NoEvent');
```

This routine requires the variable **EpL**, the current epoch, and accepts the relevant code (1 regional, 2 pca, 4 other). The QOK etc line discards the value retrieved, if the QUERY succeeded, and returns 'not success'. Here's the check for a process, or process range:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (295,
    'QUERY(SELECT process FROM PROCESS
    WHERE PROCESS.Person = $[] AND
    PROCESS.ProcType > $[] AND PROCESS.ProcType < $[]
    AND PROCESS.rEnd IS NULL)->
     QOK->COPY->BURY->NOT->SKIP->DISCARD->DIGUP',
    'IsProc');
```

We submit the person ID and range of process codes (lesser first). To identify a single code, specify the next lower and higher codes. PCA is 390, regionals are between 99 and 159. Finding current ketamine use is more tricky:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (296,
  'X->QUERY(SELECT PROCESS.process FROM RX,PROCESS WHERE
    RX.Drug = 5001 AND
    RX.Process = PROCESS.process AND
    PROCESS.Person = $[] AND
    PROCESS.rEnd IS NULL)->
    QOK->COPY->BURY->NOT->SKIP->DISCARD->DIGUP',
    'IsKetamine');
```

5.8.2 Initialisation

Let's look at the initialisation routine for the menu. Note the similarity to 903.

```
UPDATE ITEM SET iInitial =
  'NAME(comment)->
   NAME(EpL)->#1->X->&LastEpoch->SET(EpL)->
    X->&FetchIdNumber->X->&FetchSurname->
    X->&GetBed->"$[] : $[] ($[])"->TITLE'
    WHERE iID = 907;
```

Important is to see whether the patient can manage an effective cough, where appropriate. We have subsequently added 'Bowels opened'!

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (205, 1,  'Good cough',  '');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (205, 907, 205, 10,
          0.03, 0.19, 0.50, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (203, 4,  'Y',  'coY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (203, 907, 203, 11,
          0.35, 0.19, 0.07, 0.08, 1);
UPDATE ITEM SET iInitial =
       '"Cough"->&PainGet' WHERE iID = 203;
UPDATE ITEM SET iResponse =
       '"Cough"->&SetPain' WHERE iID = 203;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (204, 4,  'N',  'coN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (204, 907, 204, 12,
          0.45, 0.19, 0.07, 0.08, 1);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (209, 1,  'Bowels',  '');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (209, 907, 209, 14,
          0.58, 0.14, 0.50, 0.08, 0);
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (210, 1,  'opened',  '');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (210, 907, 210, 13,
          0.58, 0.22, 0.50, 0.08, 0);
```

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (212, 4,  'Y',  'coY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (212, 907, 212, 15,
          0.78, 0.19, 0.07, 0.08, 3);
UPDATE ITEM SET iInitial =
       '#1140->&FetchProblem' WHERE iID = 212;
UPDATE ITEM SET iResponse =
       '#1140->&SpawnProblem->#1->&SetProblem'
       WHERE iID = 212;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (213, 4,  'N',  'coN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (213, 907, 213, 16,
          0.88, 0.19, 0.07, 0.08, 3);
UPDATE ITEM SET iInitial =
       '#1140->&FetchProblem->QOK->SKIP->#1->BOOLEAN->NOT'
       WHERE iID = 213;
UPDATE ITEM SET iResponse =
       '#1140->&SpawnProblem->#0->&SetProblem'
       WHERE iID = 213;


UPDATE ITEM SET iInitial =
       '"Cough"->&PainGet->COPY->ISNULL->
       SKIP->NOT->RETURN' WHERE iID = 204;

UPDATE ITEM SET iResponse =
       'NOT->"Cough"->&SetPain' WHERE iID = 204;
```

It might still be an idea to document who the pain consultant is (if they are indeed present), but we've commented this feature out for now! Here's the code:

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (228, 1,  'Consultant',  '');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (228, 907, 228, 3,
          0.03, 0.021, 0.30, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList)
```

```
  VALUES (227, 6,  '',  'Cons',
  '->&ListConsultants');
UPDATE ITEM SET iInitial = '&FetchConsultant' WHERE iID = 227;
UPDATE ITEM SET iResponse ='&NoteConsultant' WHERE iID = 227;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (227, 907, 227, 3,
          0.02, 0.107, 0.49, 0.08, 0);
```

Here's the special function to list consultant surnames. As with all lists we provide pairs of items, the ID and the surname.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (158,
        'QMANY(SELECT PERSDATA.pdoPerson,PERSDATA.pdoSurname
        FROM PERSDATA,PERSON WHERE
        PERSDATA.pdoPerson = PERSON.person AND
        PERSON.pStatus > 7)',

        'ListConsultants');
```

. . . and here's the *NoteConsultant* routine which records who the participating consultant is! On the stack will be the unique ID of the consultant.

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(171,

'BURY->
  #1->X->&LastEpoch->BURY->
   KEY(Actor2)->DIGUP->DIGUP->
   DOSQL(INSERT INTO ACTOR2(actor2,Epoch,Arole,Person)
     VALUES($[],$[],1,$[]))',

'NoteConsultant');
```

We also need a *FetchConsultant* routine to retrieve this value. This assumes nothing more than that the current patient ID is X.

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(172,

'#1->X->&LastEpoch->
  QUERY(SELECT ACTOR2.Person FROM ACTOR2 WHERE ACTOR2.Epoch = $[])->
  QOK->SKIP->RETURN(?)->
  QUERY(SELECT PERSDATA.pdoSurname FROM PERSDATA
    WHERE PERSDATA.pdoPerson = $[])',

'FetchConsultant');
```

Technically we should probably select using MAX, but we don't anticipate many consultants flocking to one particular ward round. In addition, we might further constrain the PROCESS.

5.8.3 Pain Scores

We now turn to pain scores. Our values in the explicit list are duplicated, owing to the pairwise structure of our lists:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (206, 6,  '',  'Pm',
  '0|0|1|1|2|2|3|3|4|4|5|5|6|6|7|7|8|8|9|9|10|10|');
UPDATE ITEM SET iInitial =
    '"Movement"->&PainGet->&ReplaceNull(?)'
    WHERE iID = 206;
UPDATE ITEM SET iResponse =
    '"Movement"->&SetPain'
    WHERE iID = 206;

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (207, 6,  '',  'Pr',
  '0|0|1|1|2|2|3|3|4|4|5|5|6|6|7|7|8|8|9|9|10|10|');
UPDATE ITEM SET iInitial =
      '"Rest"->&PainGet->&ReplaceNull(?)'
      WHERE iID = 207;
UPDATE ITEM SET iResponse =
      '"Rest"->&SetPain' WHERE iID = 207;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (206, 907, 206, 2,
          0.77, 0.02, 0.18, 0.08, 0);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (207, 907, 207, 4,
          0.33, 0.02, 0.18, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (225, 1,  'PAIN: rest',  '');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (225, 907, 225, 3,
          0.02, 0.02, 0.50, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (226, 1,  'moving',  '');
```

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (226, 907, 226, 5,
          0.57, 0.02, 0.50, 0.08, 0);
```

. . . and next examine the various types of analgesic modality. *SetPain* accepts one of three pain observation types (Rest, Movement, or Cough), as well as a score, 0–10 for the first two types, and zero/one for the last. Because the score is submitted deep to the type, we swop these two values before we locate the current pain observation score (PAINSCORE). The routine assumes that the current observation ID is present in the variable EpL, created when the relevant menu was entered.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (164,
        'SWOP->$[EpL]->
        &FindPainScore->
        DOSQL(UPDATE PAINSCORE SET pso$[]=$[]
              WHERE PAINSCORE.painscore = $[])',
        'SetPain');
```

Here's *FindPainScore*, which takes the ID of an epoch on the stack and replaces it with the ID of a PAINSCORE:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (165,
        'COPY->QUERY(SELECT PAINSCORE.painscore FROM PAINSCORE
          WHERE PAINSCORE.Epoch = $[])->
        QOK->SKIP->&NewPainScore->SWOP->DISCARD->RETURN',

        'FindPainScore');
```

The subsidiary *NewPainScore* routine is simple, reading an EPOCH id, copying this, and then replacing the copy with a PAINSCORE id:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (166,
        'COPY->BURY->KEY(Painscore)->COPY->DIGUP->
        DOSQL(INSERT INTO PAINSCORE(painscore,Epoch)VALUES($[],$[]))',

        'NewPainScore');
```

PainGet retrieves a current pain score, if present, otherwise returning a question mark. In many ways it's similar to *SetPain*. Submit the relevant pain observation type, and ensure that `EpL` is set.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (167,
        '$[EpL]->&FindPainScore->
        QUERY(SELECT PAINSCORE.pso$[] FROM PAINSCORE
           WHERE PAINSCORE.painscore = $[])',

        'PainGet');
```

Some routines which invoke *PainGet* subsequently call the trivial *ReplaceNull*, which accepts a replacement value on the top of the stack, below which is the item to be tested. If the item is null, return the replacement, otherwise discard the replacement and leave the item unchanged!

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (168,
        'SWOP->COPY->ISNULL->SKIP->SWOP->DISCARD',

        'ReplaceNull');
```

We cannot use URZN as this will cause us to also replace zero (and not just null).

5.8.4 Checking analgesic modalities: regional

The user should be encouraged (but not forced) to document whether each analgesic modality is present, first regional anaesthesia. Process types which code for regional infusions have an 'ProcType' value from 200 to 250 inclusive. Underlying this should be process types between 100 and 150 inclusive, which code for the *presence of regional infusion catheters*!

In order to prevent a user from ritually ticking a process already displayed as 'present' we only reveal (or alert) when the user has already ticked! This is a safety feature, repetitively correlating the user's observations with what is present in the database! The CheckIsEvent implements this little wrinkle, using code #1 for regional procedures.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (208, 1,  'regional',  'rgn');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (208, 907, 208, 20,
          0.06, 0.341, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (217, 4,  'Y',  'reY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
```

```
          miX, miY, miW, miH, miGroup)
  VALUES (217, 907, 217, 21,
          0.08, 0.431, 0.07, 0.08, 2);
UPDATE ITEM SET iInitial = '#1->&CheckIsEvent' WHERE iID = 217;

UPDATE ITEM SET iResponse =
 '&OkComment->#104->#151->&ProcBetween->
   BOOLEAN->NOT->SKIP->MENU(REGIONAL)->
   MENU(STARTREG)'
 WHERE iID = 217;
```

The above doesn't use *FindRegional* as we wish to ignore spinals, which FindRegional identifies.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (218, 4,  'N',  'reN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (218, 907, 218, 22,
          0.18, 0.431, 0.07, 0.08, 2);
UPDATE ITEM SET iInitial = '#1->&CheckNonevent' WHERE iID = 218;
```

See how we leave both Y and N boxes unchecked until the user has committed him/herself for *this visit*! For the Y box, we look for evidence that an actual observation has been made on such a process; for the N box, we must look for a negative observation (in the NONEVENT table) related to EpL, the current observation epoch (a *general* observation, *not* an observation on an epidural 'non-process')!

Here is the 'non-event checking routine':

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (174,
    'BURY->$[EpL]->DIGUP->
     QUERY(SELECT NONEVENT.noValue FROM NONEVENT WHERE
       NONEVENT.Epoch = $[] AND
       NONEVENT.noCode = $[])->
     QOK->SKIP->RETURN(#0)->RETURN',
 'CheckNonevent');
```

We look for the relevant NONEVENT on the current observation process, EpL. Note that for a regional 'nonevent' the code is 1. If there is a recent record that no even is present, we return -1, otherwise 0. See how we use NEG to return -1 in place of 1. This little wrinkle allows us repeated access to a pushbutton which otherwise becomes inaccessible once it's on!!

Very similar is the odd CheckIsEvent:

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (240,
    'BURY->$[EpL]->DIGUP->
     QUERY(SELECT NONEVENT.noValue FROM NONEVENT WHERE
       NONEVENT.Epoch = $[] AND
       NONEVENT.noCode = $[])->
     QOK->SKIP->RETURN(#0)->
     NOT',
 'CheckIsEvent');
```

The aberration here is that *only if* the nonevent exists *and* the result is zero, i.e. an 'isevent' is recorded, do we return 1!

There's another issue, that with grouping. The standard response when we group two buttons is to *only* let the response be run when the button turns ON. If the button is already on and we click, nothing happens (we have to turn on the corresponding grouped button, thereby turning off the current one, for anything to happen on a subsequent click). But what if we wish to enter the epidural menu again because we forgot to add a particular observation?

A former solution was to 'de-group' the relevant buttons, (but then we had potential conflict when we turned off the epidural using the other button — the solution was to make the relevant changes and reload the menu!) The Perl program automatically checked for 'no associated grouping' and if so registers a -1 in place of 1, allowing a repeat click and script execution, but the initialisation script also had to set the value to -1 (NEG) rather than 1 to permit a repeat click after *intialisation*. Rather clumsy! We've now fixed the Perl code to accommodate repeated clicks.

Now let's look at the response script for the 'N' button.

```
UPDATE ITEM SET iResponse =
 '#104->#151->&HackA(#1)'
  WHERE iID = 218;
```

In the case of a regional process (codes from 105 to 150 inclusive), if the process exists (ProcBetween returns a non-zero value), we branch off to stop the regional; otherwise we record a 'non-event', i.e. that the user has documented the absence of a regional process.

Here's how we record a 'non-event', given the type of nonevent on the stack. The codes 1–4 represent regional, PCA, oral and other respectively. As usual, EpL is the current general observation (for this epoch). EpL *must* be defined.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (175,
    'COPY->$[EpL]->
```

```
    QUERY(SELECT nonevent FROM NONEVENT WHERE noCode = $[] AND Epoch = $[])->
    QOK->SKIP->=NewNonevent(#1)->
    SWOP->DISCARD->
    DOSQL(UPDATE NONEVENT SET noValue=1 WHERE nonevent = $[])',
 'SetNonevent');
```

Here's NewNonevent, which assumes the noValue value is on the top of the stack, and beneath this, noCode. Like *SetNonevent*, it requires the local variable EpL.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (241,
    'BURY->BURY->
     KEY(Nonevent)->$[EpL]->DIGUP->DIGUP->
     DOSQL(INSERT INTO NONEVENT(nonevent,Epoch,noCode,noValue)
       VALUES($[],$[],$[],$[]))',
 'NewNonevent');
```

Next we have the very similar SetEvent, which records user documentation of an event (curiously by using a 0 in the NONEVENT noValue field).

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (242,
    'COPY->$[EpL]->
    QUERY(SELECT nonevent FROM NONEVENT WHERE noCode = $[] AND Epoch = $[])->
    QOK->SKIP->=NewNonevent(#0)->
    SWOP->DISCARD->
    DOSQL(UPDATE NONEVENT SET noValue=0 WHERE nonevent = $[])',
 'SetEvent');
```

By the way, here's our generic warning/error alert, which will often be invoked by =Fail rather than &Fail, to prevent further processing of the script!

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (176,
    'Alert($[])',
 'Fail');
```

A similar but slower and more dramatic routine is *FailAndReload* which not only 'fails' but reloads the current menu.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (195,
    'Alert( $[] )->MENU(0)',
 'FailAndReload');
```

KillManyProcs accepts the patient ID, the lower bound of the process nature, and the upper bound and terminates all processes with a nature *between* those bounds.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (205,
    'BURY->BURY->BURY->NOW->DIGUP->DIGUP->DIGUP->
    DOSQL(UPDATE PROCESS SET rEnd=TIMESTAMP ''$[]'' WHERE
      PROCESS.Person = $[] AND
      PROCESS.rEnd IS NULL AND
      PROCESS.ProcType > $[] AND
      PROCESS.ProcType < $[])',
 'KillManyProcs');
```

Here's the routine to do the dastardly deed . . . *Killproc.* Given the process ID, terminate it. We use DEPTH to check whether there's anything on the stack — if not, we stop (possibly also terminating a REPEAT statement in the caller).

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (182,
    'DEPTH->GREATER(#0)->SKIP->STOP->
    BURY->NOW->DIGUP->
    DOSQL(UPDATE PROCESS SET rEnd=TIMESTAMP ''$[]''
      WHERE process = $[])',
 'KillProc');
```

Finally, *KillDated*, a variant of *KillManyProcs* which requires the variable **rdate** as the termination date stamp. This isn't very precise, always specifying a time of 00:00:00.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (288,
    'BURY->BURY->BURY->$[rdate]->"$[] 00:00:00"->
     DIGUP->DIGUP->DIGUP->
    DOSQL(UPDATE PROCESS SET rEnd=TIMESTAMP ''$[]'' WHERE
      PROCESS.Person = $[] AND
      PROCESS.rEnd IS NULL AND
      PROCESS.ProcType > $[] AND
      PROCESS.ProcType < $[])',
 'KillDated');
```

We initially made rather a mess of things by leaving out 'PROCESS.rEnd IS NULL', forcing prior processes (correctly terminated) to overlap with the current one, if multiple processes of this type existed! We only fixed this on 31/1/2008.

5.8.5 Checking for IV PCA

Next let's do intravenous PCA. In a manner analogous to our plan for regionals, we do not reveal the process until the user has! We use the event code for IV PCA as #2.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (214, 1,  'IV PCA',  'ivpc');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (214, 907, 214, 23,
          0.34, 0.341, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (219, 4,  'Y',  'ivY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (219, 907, 219, 24,
          0.32, 0.431, 0.07, 0.08, 4);
UPDATE ITEM SET iInitial =
    '#2->&CheckIsEvent' WHERE iID = 219;
UPDATE ITEM SET iResponse = '&OkComment->&GoIvPca' WHERE iID = 219;



INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (220, 4,  'N',  'ivN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (220, 907, 220, 25,
          0.42, 0.431, 0.07, 0.08, 4);
UPDATE ITEM SET iInitial =
       '#2->&CheckNonevent' WHERE iID = 220;
```

Here's the PCA 'N' button response. Note that the process code for IV PCA is 390.

```
UPDATE ITEM SET iResponse =
 '#389->#391->&HackA(#2)'
  WHERE iID = 220;
```

5.8.6 Checking for orals

Next oral therapy, which is similar to regionals and also to PCA . . . but the process code for enteral drugs is 50 and the event code is #3.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (211, 1,  'orals',  'ora');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (211, 907, 211, 26,
          0.59, 0.341, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (221, 4,  'Y',  'orY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (221, 907, 221, 27,
          0.56, 0.431, 0.07, 0.08, 6);
UPDATE ITEM SET iInitial =
       '#3->&CheckIsEvent'
       WHERE iID = 221;
UPDATE ITEM SET iResponse =
       '&OkComment->&GoOrals' WHERE iID = 221;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (222, 4,  'N',  'orN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (222, 907, 222, 28,
          0.66, 0.431, 0.07, 0.08, 6);
UPDATE ITEM SET iInitial =
       '#3->&CheckNonevent' WHERE iID = 222;

UPDATE ITEM SET iResponse =
 '#49->#51->&HackA(#3)'
  WHERE iID = 222;
```

5.8.7 Checking for other analgesic modalities

. . . and last on the line is a mishmash of other modalities, analogous to the above. The event codes we use (as per *AnalgesiaDBpart1*) range from 260 to 299 inclusive, and encompass plain old ‘IV drug administration’, which is distinct from IV PCA (PCA codes are in the range 300–399). The event code is #4.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (215, 1,  'other',  'oth');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (215, 907, 215, 29,
          0.81, 0.341, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
```

```
          VALUES (223, 4,  'Y',  'otY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (223, 907, 223, 30,
          0.80, 0.431, 0.07, 0.08, 8);
UPDATE ITEM SET iInitial =
       '#4->&CheckIsEvent' WHERE iID = 223;
UPDATE ITEM SET iResponse =
      '&OkComment->&GoOther' WHERE iID = 223;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (224, 4,  'N',  'otN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (224, 907, 224, 31,
          0.90, 0.431, 0.07, 0.08, 8);
UPDATE ITEM SET iInitial =
       '#4->&CheckNonevent' WHERE iID = 224;
UPDATE ITEM SET iResponse =
       'ALERT(Check/record OTHER RX non-event)' WHERE iID = 224;

UPDATE ITEM SET iResponse =
 '#259->#300->&HackA(#4)'
  WHERE iID = 224;
```

The following item will pull in the most recent comment.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (159, 1,  '-',  're');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (159, 907, 159, 3,
          0.01, 0.780, 0.98, 0.08, 1);
```

. . . and here is the clumsy initialisation routine:

```
UPDATE ITEM SET iInitial =
   'X->QMANY(SELECT MAX(COMMENT.comment) FROM COMMENT,EPOCH,PROCESS
   WHERE COMMENT.Epoch = EPOCH.epoch AND EPOCH.Process = PROCESS.process
   AND PROCESS.Person = $[])->
   QOK->SKIP->STOP->
   QUERY(SELECT COMMENT.cText FROM COMMENT WHERE COMMENT.comment = $[])'
 WHERE iID = 159;

UPDATE ITEM SET iResponse =
  'X->QMANY(SELECT MAX(COMMENT.comment) FROM COMMENT,EPOCH,PROCESS
   WHERE COMMENT.Epoch = EPOCH.epoch AND EPOCH.Process = PROCESS.process
```

```
   AND PROCESS.Person = $[])->
   QOK->SKIP->STOP->
   QUERY(SELECT COMMENT.cText FROM COMMENT WHERE COMMENT.comment = $[])->Alert'
  WHERE iID =159;
```

We attach our 'problem' epoch to the general data observation process (code 1), a useful economy!

We provide the facility to add a comment:

```
-- comment box:
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1663, 907, 661, 3,
          0.001, 0.570, 0.990, 0.08, 0);
-- 'Add' button:
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1664, 907, 662, 4,
          0.820, 0.660, 0.170, 0.08, 0);

--INSERT INTO ITEM (iID, iType, iText, iName)
--          VALUES (1665, 1,  'Enter new comment, click =>',  '');
--INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
--          miX, miY, miW, miH, miGroup)
--  VALUES (1665, 907, 1665, 14,
--          0.010, 0.660, 0.800, 0.08, 0);

-- comment button:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (158, 2,  'All comments',  'AdCmt', '',  1);

UPDATE ITEM SET iResponse = '&OkComment->MENU(COMMENTS)' WHERE iID = 158;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (730, 907, 158, 40,
          0.28, 0.90, 0.44, 0.08, 0);
```

5.9 Adding operation data (908)

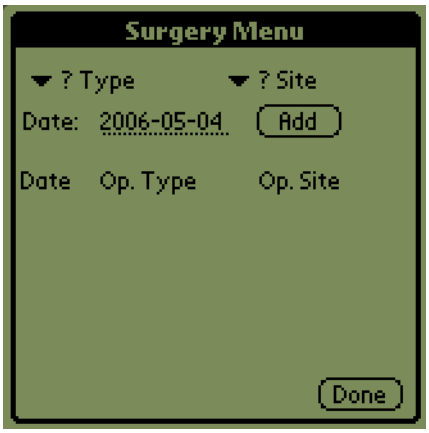


Figure 9: Adding an operation

Previously in this section we coded both surgical type and site. We have amended this to code for nature of surgery (e.g. hepatobiliary) but have removed the 'site' specification for several reasons the main ones being both redundancy (obstetric operation on lower abdomen) and the potential for all sorts of nonsense (eye surgery on the foot?)!

Here we display observations on surgical site and type. First we have to find all surgical processes on this patient. For each process we find an observation with an associated SURGTYPEOB and this allows us to list the details.[56] Here's the idea. The process ID for an operation is 500:

```
SELECT EPOCH.epoch FROM EPOCH,PROCESS
  WHERE EPOCH.Process = PROCESS.process AND
  PROCESS.ProcType = 500 AND
  PROCESS.Person = $[]
```

We create a polymorphic table along the same lines of our comment table (Section 5.7).

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
         VALUES (620, 8,  '[No operation]',  'Cmnt', '',  6);
```

Here we initialise the table 'V items' so we can populate its rows:

[56] Although its conceivable that we might have multiple observations and multiple types and sites for one such observation, we will initially cheat and assume one per epoch, and one epoch per procedure. Nasty :-(

```
UPDATE ITEM SET iInitial = 'X->QMANY(SELECT EPOCH.epoch
    FROM EPOCH,PROCESS
    WHERE EPOCH.Process = PROCESS.process AND
    PROCESS.ProcType = 500 AND
    PROCESS.Person = $[])'
  WHERE iID = 620;

--INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
--          VALUES (623, 1,  '-',  'Sit1', '',  1);
-- [site coding disabled]
--
--INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
--            irName, irFraction)
--    VALUES (623, 620, 623, 3,
--            'Op. Site', 0.40);
--- site of surgery (disabled)

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
            irName, irFraction)
    VALUES (621, 620, 621, 1,
            'Date', 0.24);
--- date of observation

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
            irName, irFraction)
    VALUES (622, 620, 622, 2,
           'Op. Type', 0.34);
--- type of surgery

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
            irName, irFraction)
    VALUES (624, 620, 624, 3,
            'Note', 0.42);


UPDATE ITEM SET iInitial =
   'V->QUERY(SELECT SURGTYPEOB.SurgType FROM SURGTYPEOB
       WHERE SURGTYPEOB.Epoch = $[])->
    COPY->ISNULL->NOT->SKIP->RETURN(?)->
      QUERY(SELECT SURGTYPE.ctText FROM SURGTYPE
      WHERE SURGTYPE.surgtype = $[])'
   WHERE iID =622;
```

We've now disabled site coding. Here was the script:

```
UPDATE ITEM SET iInitial =
       'V->QUERY(SELECT SURGSITEOB.SurgSite FROM SURGSITEOB
          WHERE SURGSITEOB.Epoch = $[])->
```

```
    COPY->ISNULL->NOT->SKIP->RETURN(?)->
          QUERY(SELECT SURGSITE.csText FROM SURGSITE
          WHERE SURGSITE.surgsite = $[])'
       WHERE iID =623;
```

Let's get the date associated with a procedure:

```
UPDATE ITEM SET iInitial =
       'V->QUERY(SELECT PROCESS.rStart FROM EPOCH,PROCESS
       WHERE EPOCH.epoch = $[] AND EPOCH.Process = PROCESS.process)->
       SPLIT( )->DISCARD->&ShortDate'
       WHERE iID =621;
```

. . . and the comment:

```
UPDATE ITEM SET iInitial =
       'V->QUERY(SELECT cText FROM COMMENT
       WHERE Epoch = $[])'
       WHERE iID =624;
```

We make our operation comment clickable:[57]

```
UPDATE ITEM SET iResponse =
       'V->QUERY(SELECT cText FROM COMMENT
           WHERE Epoch = $[])->Alert'
       WHERE iID =624;
```

Next, we have the actual operation *menu*, which includes the above table:

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (908, 20, 'Surgery', 'SURGERY');
```

Here's the initialisation string:

```
UPDATE ITEM SET iInitial = 'NAME(typeId)->NAME(surgDate)->
    NAME(surgnote)->NULL->SET(surgnote)->
    #0->SET(typeId)->
    NAME(id)->X->SET(id)->
    NOW->set(surgDate)->
    X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
  WHERE iID = 908;
```

We set all values to zero for easy testing later on! The variable *siteId* has been removed from the above initialisation (9/2007).

[57] A peculiar convention/hack is that in order to make text fields clickable, the miOrder must be nonzero in the containing MENUITEMS row defining the table containing the comment!

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (630, 908, 908, 0,
            0.001, 0.001, 0.990, 0.990, 0);
--- Self-reference.

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
   miX, miY, miW, miH, miGroup, miEnabled)
  VALUES (631, 908, 620, 22,
   0.001, 0.300, 0.999, 0.500, 1, 0);
-- we make the miGroup nonzero to allow 'clickable' comments on the PDA!

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
         VALUES (632, 2,  'Done',  'done', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (632, 908, 632, 10,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
```

Here's the response to quitting the menu. If both selectable items are unfilled (null surgnote, zero typeId), then leaving is fine; otherwise confirm before abandoning data!

```
UPDATE ITEM SET iResponse =
  '$[surgnote]->ISNULL->$[typeId]->SAME(#0)->AND->NOT->SKIP->MENU(-1)->
  CONFIRM(Abandon data?)->NOT->SKIP->MENU(-1)'
  WHERE iID = 632;
```

We continue..

```
INSERT INTO ITEM (iID, iType, iText, iName)
         VALUES (637, 1,  'on:',  'sdat');

INSERT INTO ITEM (iID, iType, iText, iName, iLines)
         VALUES (639, 10,  '',  'c', 1);

UPDATE ITEM SET iResponse = 'SET(surgnote)'
    WHERE iID = 639;

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
         VALUES (633, 12,  '',  'ctxt', '',  1);
UPDATE ITEM SET iInitial =
       'NOW->SPLIT( )->DISCARD' WHERE iID = 633;
-- note that iType is 12 (date field) for date picker!

INSERT INTO ITEM (iID, iType, iText, iName)
```

```
          VALUES (638, 1,  'note:',  'sdat');


UPDATE ITEM SET iResponse =
       '"$[] 00:00:00"->SET(surgDate)'
       WHERE iID = 633;
-- above should first VALIDate ?!
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (634, 2,  'Add',  'cbtn', '',  1);
UPDATE ITEM SET iResponse =  '&InsertOpData'
  WHERE iID = 634;
-- if MakeOperation succeeds, it reloads the menu!


-- INSERT INTO ITEM (iID, iType, iText, iName, iList)
--  VALUES (635, 6,  '',  'Op. Site', '->&ListOpSites');
-- UPDATE ITEM SET iInitial = '"? Site"' WHERE iID = 635;
```

Site capture is disabled:

```
{\footnotesize\begin{verbatim}
UPDATE ITEM SET iResponse =
       'SET(siteId)'
   WHERE iID = 635;


INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (636, 6,  '',  'Op. Type', '->&ListOpTypes');
UPDATE ITEM SET iInitial = '"? Type"' WHERE iID = 636;

UPDATE ITEM SET iResponse =
       'SET(typeId)'
    WHERE iID = 636;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (633, 908, 633, 3,
          0.63, 0.05, 0.35, 0.08, 0);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (637, 908, 637, 3,
          0.50, 0.05, 0.06, 0.08, 0);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (638, 908, 638, 3,
```

```
          0.01, 0.15, 0.10, 0.08, 0);
-- comment label 'note:'

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (639, 908, 639, 3,
          0.16, 0.15, 0.66, 0.08, 0);
-- actual comment box (entry) -- short!

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (636, 908, 636, 3,
          0.01, 0.050, 0.48, 0.08, 0);
--INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
--          miX, miY, miW, miH, miGroup)
--    VALUES (635, 908, 635, 3,
--          0.50, 0.050, 0.48, 0.08, 0);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (634, 908, 634, 4,
          0.83, 0.150, 0.15, 0.08, 0);
```

Here are the functions to list operation sites and types. As usual, they return *pairs* of items to use within a poplist.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (159,
        'QMANY(SELECT surgtype,ctText FROM SURGTYPE)',
        'ListOpTypes');
```

Site coding is disabled:

```
{\footnotesize\begin{verbatim}
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (160,
        'QMANY(SELECT surgsite,csText FROM SURGSITE)',
        'ListOpSites');
```

And finally, the all-important insertion of the actual (operation type) details! The 'unfilled' typeId value is zero and causes an error message; a comment (surgnote) is optional. If no comment exits, the MENU(0) statement terminates the routine and reloads the menu.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (161,
    '$[typeId]->SAME(#0)->NOT->SKIP->=FailAndReload(Op type?)->
```

```
$[surgDate]->#500->&DatedPandE->COPY->
BURY->BURY->
KEY(Surgtypeob)->$[typeId]->DIGUP->
DOSQL(INSERT INTO SURGTYPEOB(surgtypeob,SurgType,Epoch)
  VALUES($[],$[],$[]))->
$[surgnote]->ISNULL->NOT->SKIP->MENU(0)->
KEY(Comment)->DIGUP->$[surgnote]->&FixSQL->
DOSQL(INSERT INTO COMMENT(comment,Epoch,cText)
  VALUES($[],$[],''$[]''))->
 MENU(0)',
   'InsertOpData');
```

The attached comment is optional. *DatedPandE* creates the epoch and process given the relevant date. Type 500 is a surgical process. Formerly we used ProcAndEpoch which doesn't accommodate a start date different from today (now)! See the comment on MARK and URZN at MakeGeneralComment. We don't go to the finicky detail of adding in the operation time.

Here's *DatedPandE*, which allows us to create *now* a process with the start timestamp supplied on the stack below the process type code:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (238,
    'BURY->BURY->"Process"->KEY->copy->$[id]->DIGUP->now->me->DIGUP->
    DOSQL(INSERT INTO PROCESS
     (process,Person,rStart,rCreated,rPlanner,ProcType)
     VALUES($[],$[],TIMESTAMP ''$[]'',TIMESTAMP ''$[]'',$[],$[]))->
    "Epoch"->KEY->copy->BURY->SWOP->now->me->
    DOSQL(INSERT INTO EPOCH(epoch,Process,oMade,Person)
    VALUES($[],$[],TIMESTAMP ''$[]'',$[]))->DIGUP',
   'DatedPandE');
```

5.10 The regional menu (904)

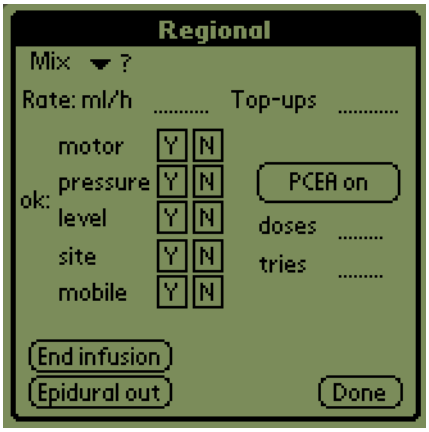


Figure 10: Regional menu

First we create the main menu.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (904, 20, 'Regional', 'REGIONAL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH, miGroup)
               VALUES (904, 904, 904, 12,
                       0.000, 0.200, 0.999, 0.999, 0);
```

Of particular importance is the menu initialisation in which we create several local variables including EpL (the most recent epoch) to which we attach a signal that we've recently entered this menu (SetEvent), and regional-related processes and epochs (prR, EpR). We also have similar variables for *infusion*-related processes and epochs — prQ and EpQ.

In the following the local variable 'fentanyl' is not used, but we still must set it up, as within the menu we use GetInfusionLabel which tests for the presence of fentanyl and requires that variable!

```
UPDATE ITEM SET iInitial =
  'NAME(EpL)->NAME(EpR)->NAME(prR)->NAME(prQ)->NAME(EpQ)->
    NAME(fentanyl)->
    #1->X->&LastEpoch->SET(EpL)->
    &SetEvent(#1)->
    &FindRegional->SET(prR)->
    $[prR]->
    QUERY(SELECT ProcType FROM PROCESS WHERE process = $[])->
    QUERY(SELECT rptNature FROM PROCTYPE WHERE proctype = $[])->
```

```
X->&FetchIdNumber->SWOP->
TITLE($[]: $[])->
&NewRgnEpoch->SET(EpR)->
&CheckInfusion->SET(prQ)->
&FindInfuObs->SET(EpQ)->
NAME(pethidine)->#0->SET(pethidine)->
$[prQ]->QUERY(SELECT RX.Drug FROM RX WHERE RX.Process=$[])->
QOK->SKIP->RETURN->SAME(#103)->SET(pethidine)'
WHERE iID = 904;
```

The final section where we define the NAME pethidine will set this value to 1 if a pethidine infusion (code 103) is being administered.[58] This allows us to configure later items in the menu!

Check out NewRgnEpoch in particular — it requires that EpL is defined, as this value is used as a 'benchmark' for determining whether a *recent* process exists!

On entering the REGIONAL menu, we first identify the most recent general observation, and then find/create a new, specific regional observation. We place this in the variable EpR, and reference it throughout the menu. We also set aside the important local variable prR, which is used to store the regional *process*.[59]

The additional local variables `prQ` and `EpQ` are respectively used to describe the actual process of epidural infusion, and the current epoch on this process (See Section 5.10.2).

FindRegional locates the current regional process.[60] Once the process is established, we find/create a new epoch in that process, and retain this epoch.

We only create e.g. the epidural catheter access process (code 110) and NOT at this stage the infusion process (e.g. code 210 for epidural infusion, code 310 for PCEA), because we haven't been told whether an infusion is occurring or not! (The user does this later). However, we still look for an existing infusion, and a recent epoch on this infusion, using *FindAdministration* and *FindInfuObs*! This approach caters for the case where we *re-enter* the menu, and want to see what we wrote! These last two functions return a value of zero if they fail.

AN ASIDE: Because observations such as motor power, pressure areas, block and site are not necessarily dependent on the infusion,[61] these observations will be recorded on the epoch associated with the epidural process and not the infusion process!

```
INSERT INTO FUN (fKey, fBody, fName)
```

[58] This 'hard coding' is rather awful. Ideally we should look up the code for epidural pethidine.

[59] A bit of a hack!

[60] At present the assumption is that only one such process is current. The database design allows for more but our implementation is more 'reasonable'.

[61] There might be cord trauma or an epidural haematoma!

```
    VALUES (179,
    '#99->#151->&ProcBetween->
     QOK->SKIP->MENU(#1)->RETURN',

 'FindRegional');
```

Formerly we had the first parameter as #98, but this would include follow-up observations, an obvious error which caused some grief.[62] Unfortunately when we first implemented this, we ignored the use of *FindRegional* in menu 930, which caused a fatal (but interesting) PDA error!

FindRegional is used in the initialisation of menus 904, 930 (regional check at 24 hours) and 952 (Overview of regional problems). A failure is fairly catastrophic, so we force a MENU(#1) if the SQL fails!

It does *not* include epidural checks at 24 hours (code 99). *ProcBetween* locates any *active* process inside the range provided (for the patient X). It returns zero if the process doesn't exist.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (177,
    'BURY->BURY->X->DIGUP->DIGUP->
     QUERY(SELECT PROCESS.process FROM PROCESS WHERE
       PROCESS.Person = $[] AND
       PROCESS.rEnd IS NULL AND
       PROCESS.ProcType > $[] AND
       PROCESS.ProcType < $[])->
     QOK->SKIP->RETURN(#0)->RETURN',
 'ProcBetween');
```

Some callers of *ProcBetween* rely on QOK as a subsequent test.

Given a process ID in prR, *NewRgnEpoch* finds a recent epoch on that process, or if one doesn't exist, makes one, returning the code. The process ID *must* already have been stored in the variable prR!

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (181,
    '$[prR]->&RecentProcObs->COPY->#0->SAME->SKIP->RETURN->
     DISCARD->KEY(Epoch)->COPY->NOW->ME->$[prR]->
     DOSQL(INSERT INTO EPOCH(epoch,oMade,Person,Process)
        VALUES($[],TIMESTAMP ''$[]'',$[],$[]))->
     COPY->BURY->KEY(Rgnobs)->DIGUP->
     DOSQL(INSERT INTO RGNOBS(rgnobs,Epoch)VALUES($[],$[]))',

        'NewRgnEpoch');
```

[62] This error did suggest that if we have put in an epidural, stop it, and then start another *epidural*, we might remove the type 99 process!

In addition, we make a RGNOBS table entry for recording of pressure/site/motor block and sensory level observations. The above routine is rather clumsy.[63]

RecentProcObs is similar to *LastEpoch* but here we simply specify the ID of the process (not the type) and find an epoch, if present. If there is *no* matching epoch we immediately return zero, otherwise we ensure that the epoch is *greater than* EpL, the general epoch we pulled out previously! Obviously, EpL must exist for this to work.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (180,
    'QMANY(SELECT MAX(EPOCH.epoch) FROM EPOCH
     WHERE EPOCH.Process = $[])->
    QOK->SKIP->RETURN(#0)->COPY->$[EpL]->
    GREATER->SKIP->RETURN(#0)->RETURN',

        'RecentProcObs');
```

5.10.1 Filling in menu details

Let's examine the menu components. Here is the 'Done' button, followed by the pushbuttons for the five options of motor block (OK or not), pressure areas, mobility, level and site.

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (401, 2,  'Done',  'Exitbtn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (401, 904, 401, 39,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
UPDATE ITEM SET iResponse = 'MENU(-1)'
  WHERE iID = 401;

-- motor:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (402, 4,  'Y',  'moY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (402, 904, 402, 8,
          0.35, 0.23, 0.07, 0.08, 1);
UPDATE ITEM SET iInitial = '"Motor"->$[EpR]->&FetchEpi' WHERE iID = 402;
UPDATE ITEM SET iResponse =
   'SKIP->RETURN->#1->&RecordEpi(Motor)'
   WHERE iID = 402;
```

[63] Might we not always return the key into the RGNOBS table?

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (403, 4,  'N',  'moN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (403, 904, 403, 9,
          0.44, 0.23, 0.07, 0.08, 1);
UPDATE ITEM SET iInitial =
       '"Motor"->$[EpR]->&FetchEpi->NOT'
       WHERE iID = 403;
UPDATE ITEM SET iResponse =
       'SKIP->RETURN->#0->&RecordEpi(Motor)'
       WHERE iID = 403;
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (404, 1,  'motor',  'moL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (404, 904, 404, 7,
          0.10, 0.23, 0.07, 0.08, 0);

-- pressure areas
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (405, 4,  'Y',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (405, 904, 405, 12,
          0.35, 0.33, 0.07, 0.08, 2);
UPDATE ITEM SET iInitial =
       '"Pressure"->$[EpR]->&FetchEpi' WHERE iID = 405;
UPDATE ITEM SET iResponse =
       'SKIP->RETURN->#1->&RecordEpi(Pressure)'
       WHERE iID = 405;
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (406, 4,  'N',  'prN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (406, 904, 406, 13,
          0.44, 0.33, 0.07, 0.08, 2);
UPDATE ITEM SET iInitial =
       '"Pressure"->$[EpR]->&FetchEpi->NOT' WHERE iID = 406;
UPDATE ITEM SET iResponse =
       'SKIP->RETURN->#0->&RecordEpi(Pressure)'
       WHERE iID = 406;
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (407, 1,  'pressure',  'prL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (407, 904, 407, 14,
          0.10, 0.33, 0.07, 0.08, 0);
```

```
-- level
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (408, 4,  'Y',  'blY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (408, 904, 408, 15,
          0.35, 0.43, 0.07, 0.08, 3);
UPDATE ITEM SET iInitial =
       '"Level"->$[EpR]->&FetchEpi' WHERE iID = 408;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#1->&RecordEpi(Level)'
      WHERE iID = 408;
-- 1 signals GOOD sensory block
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (409, 4,  'N',  'blN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (409, 904, 409, 16,
          0.44, 0.43, 0.07, 0.08, 3);
UPDATE ITEM SET iInitial =
       '"Level"->$[EpR]->&FetchEpi->NOT' WHERE iID = 409;
UPDATE ITEM SET iResponse =
       'SKIP->RETURN->#0->&RecordEpi(Level)'
       WHERE iID = 409;
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (410, 1,  'level',  'blL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (410, 904, 410, 17,
          0.10, 0.43, 0.07, 0.08, 0);
-- 0 signals inadequate sensory block

-- site
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (411, 4,  'Y',  'siY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (411, 904, 411, 18,
          0.35, 0.53, 0.07, 0.08, 4);
UPDATE ITEM SET iInitial =
       '"Site"->$[EpR]->&FetchEpi' WHERE iID = 411;
UPDATE ITEM SET iResponse =
       'SKIP->RETURN->#1->&RecordEpi(Site)'
       WHERE iID = 411;
-- 1 signals 'site OK'
INSERT INTO ITEM (iID, iType, iText, iName)
```

```
          VALUES (412, 4,  'N',  'siN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (412, 904, 412, 19,
          0.44, 0.53, 0.07, 0.08, 4);
UPDATE ITEM SET iInitial =
       '"Site"->$[EpR]->&FetchEpi->NOT' WHERE iID = 412;
UPDATE ITEM SET iResponse =
       'SKIP->RETURN->#0->&RecordEpi(Site)' WHERE iID = 412;
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (413, 1,  'site',  'siL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (413, 904, 413, 20,
          0.10, 0.53, 0.07, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (414, 1,  'ok:',  'okL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (414, 904, 414, 3,
          0.001, 0.380, 0.13, 0.08, 7);

--mobility:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (428, 4,  'Y',  'mbY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (428, 904, 428, 23,
          0.35, 0.63, 0.07, 0.08, 5);
UPDATE ITEM SET iInitial =
       '"Mobile"->$[EpR]->&FetchEpi' WHERE iID = 428;
UPDATE ITEM SET iResponse =
       'SKIP->RETURN->#1->&RecordEpi(Mobile)'
       WHERE iID = 428;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (429, 4,  'N',  'mbN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (429, 904, 429, 3,
          0.44, 0.63, 0.07, 0.08, 5);
UPDATE ITEM SET iInitial =
       '"Mobile"->$[EpR]->&FetchEpi->NOT' WHERE iID = 429;
UPDATE ITEM SET iResponse =
       'SKIP->RETURN->#0->&RecordEpi(Mobile)'
       WHERE iID = 429;
INSERT INTO ITEM (iID, iType, iText, iName)
```

```
          VALUES (430, 1,  'mobile',  'mbL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (430, 904, 430, 24,
          0.10, 0.63, 0.07, 0.08, 0);
```

Here's *FetchEpi*:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (183,
    'QUERY(SELECT rgo$[] FROM RGNOBS WHERE Epoch = $[])->
     QOK->NOT->SKIP->RETURN->NULL',
        'FetchEpi');
```

Using the current epoch on the (epidural) insertion process, EpR, we obtain the relevant epoch (Pressure, Site, Motor, or Level), and display it. Note the reciprocal relationship between the Y and N buttons, which ideally should be implemented by querying the state just once.[64] The latter part (QOK) returns NULL even if the SQL query failed.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (184,
    'SWOP->$[EpR]->
     DOSQL(UPDATE RGNOBS SET rgo$[]=$[] WHERE Epoch = $[])',
        'RecordEpi');
```

5.10.2 Details of the epidural infusion process

Next, numeric boxes (with their labels). We might later replace these with 'custom widgets' to facilitate easy numeric entry, but for now we pop up the system keyboard! (iType = code 14). The PCEA process code is 310.

```
-- pcea good:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (2421, 1,  'doses',  'gd', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (2421, 904, 2421, 25,
          0.55, 0.45, 0.200, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (422, 14,  '',  'goodT', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
```

[64] We haven't implemented messaging between buttons in the current version of the program.

```
    VALUES (422, 904, 422, 26,
          0.80, 0.45, 0.12, 0.08, 0);
UPDATE ITEM SET iResponse =
       'INTEGER->"Good"->$[EpQ]->&PcaRecord'
       WHERE iID = 422;
UPDATE ITEM SET iInitial =
       '$[pethidine]->NOT->SKIP->FAIL->$[prQ]->&pcAble->"Good"->$[EpQ]->&GetPca'
       WHERE iID = 422;

-- pcea tries:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (419, 1,  'tries',  'PtSrn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (419, 904, 419, 27,
          0.60, 0.55, 0.200, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (420, 14,  '',  'tryT', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (420, 904, 420, 28,
          0.80, 0.55, 0.12, 0.08, 0);
UPDATE ITEM SET iResponse =
       'INTEGER->"Tries"->$[EpQ]->&PcaRecord'
       WHERE iID = 420;
UPDATE ITEM SET iInitial =
       '$[prQ]->&pcAble->"Tries"->$[EpQ]->&GetPca'
       WHERE iID = 420;

-- top-ups:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (417, 1,  'Top-ups',  'PtSrn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (417, 904, 417, 5,
          0.53, 0.11, 0.200, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (418, 14,  '',  'topuT', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (418, 904, 418, 6,
          0.80, 0.11, 0.15, 0.08, 0);
UPDATE ITEM SET iResponse =
       'INTEGER->&TopupSet' WHERE iID = 418;
UPDATE ITEM SET iInitial = '&FindTopups' WHERE iID = 418;
```

```
-- infusion rate:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (415, 1,  'Rate: ml/h',  'PtSrn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (415, 904, 415, 3,
          0.01, 0.11, 0.200, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (416, 14,  '',  'RateT', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (416, 904, 416, 4,
          0.33, 0.11, 0.15, 0.08, 0);
UPDATE ITEM SET iResponse =
       'INTEGER->&InfuRateSet' WHERE iID = 416;
UPDATE ITEM SET iInitial = '&FindInfuRate' WHERE iID = 416;
```

The following is customised to allow alterations if epidural pethidine is selected as being infused:

```
UPDATE ITEM SET iInitial =
'$[pethidine]->SKIP->RETURN(   doses)->"dose(mg):"'
  WHERE iID = 2421;
```

In addition we have the alternative text box for the total dose (mg) value:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (2422, 14,  '',  'goodT', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (2422, 904, 2422, 3,
          0.83, 0.45, 0.12, 0.08, 0);
UPDATE ITEM SET iResponse = 'INTEGER->&SetTotal'
       WHERE iID = 2422;
UPDATE ITEM SET iInitial =
       '$[pethidine]->SKIP->FAIL->$[prQ]->&pcAble->$[EpQ]->&FindTotal'
       WHERE iID = 2422;
```

If the infusion *is* pethidine, then only is the box created! The nature of the epoch is quite different, for here we must use the RXOBS table to record the total dose.

For the mix, we extract the options from the database. The process code for epidural infusion is 210: we format the following to accommodate all infusions (currently up to code 250).

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (423, 1,  'Mix',  '');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (423, 904, 423, 1,
          0.03, 0.001, 0.15, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (424, 6,  '',  'rgnli', '->#1->&ByFormulation');
--- epidural=1;
UPDATE ITEM SET iInitial =
       '&GetInfusionLabel' WHERE iID = 424;
UPDATE ITEM SET iResponse =
       '&EpidInfuSet' WHERE iID = 424;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (424, 904, 424, 2,
          0.16, 0.01, 0.82, 0.08, 0);
```

Determining the infusion is mildly complex. When the user selects a particular infusion (say 'standard mix'), the program first looks for an epidural infusion (with or without PCEA) in existence. If the infusion isn't documented as present, then it's created. Otherwise the documented RX (see that table) is checked to see whether the selected infusion is the same — if things have changed, then the user must confirm this, and the program will then *stop* the current infusion process and establish a new one as specified. Conversely, if things are as before, we simply store the selected process in the local variable `prQ`.

Next, we have to either obtain (if it exists) or create an epoch associated with the infusion process. This value is then stored in the local variable `EpQ`.

If the user tries to enter a rate or other observation on the infusion process before selecting the type of infusion, they should be gently asked to specify the type of infusion first.[65]

FindAdministration doesn't try to make a new infusion, merely locating an old but active one. At present we limit this to an epidural infusion. This routine takes three arguments off the stack — the patient, the lower process code, and the upper one.

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (186,
```

[65] This may cause some irritation, but it's pretty awful practice not to even check the drug being infused!

```
    'QUERY(SELECT PROCESS.process FROM PROCESS
       WHERE PROCESS.Person = $[]
       AND PROCESS.rEnd IS NULL AND PROCESS.ProcType > $[]
       AND PROCESS.ProcType < $[])->
       QOK->SKIP->RETURN(#0)->RETURN',
  'FindAdministration');
```

CheckInfusion checks for both an ordinary regional infusion process and a PCA regional infusion, returning either zero or the process code for the infusion. We submit nothing on the stack!

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (248,
    'X->#209->#251->&FindAdministration->
     COPY->SAME(#0)->SKIP->RETURN->DISCARD->
     X->#309->#351->&FindAdministration',
  'CheckInfusion');
```

Next, *FindInfuObs*. Iff an infusion epoch exists which is greater than the local variable `EpL`, then we return it, otherwise zero. We assume that the current infusion process has already been stored in `prQ`.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (187,
    '$[EpL]->$[prQ]->
     QUERY(SELECT EPOCH.epoch FROM EPOCH WHERE
       EPOCH.epoch > $[] AND
       EPOCH.Process = $[])->
     QOK->SKIP->RETURN(#0)->RETURN',

   'FindInfuObs');
```

FindInfuRate checks the relevant (non-zero) epoch for a rate value, and that's it.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (188,
    '$[EpQ]->
     QMANY(SELECT MAX(INFUSIONOBS.infusionobs) FROM INFUSIONOBS WHERE
       INFUSIONOBS.Epoch = $[])->
     QOK->SKIP->RETURN->
     QUERY(SELECT INFUSIONOBS.inoRate FROM INFUSIONOBS WHERE
       INFUSIONOBS.infusionobs = $[])->
     DIV(#1000)',

   'FindInfuRate');
```

We allow for multiple INFUSIONOBS on one epoch (if we're fiddling), selecting the most recent! We also convert from microlitres per hour to ml per hour, using integer arithmetic. The converse is *InfuRateSet*.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (189,
    '$[EpQ]->SAME(#0)->NOT->SKIP->
    =FailAndReload(Please select infusion Mix!)->
    COPY->ISNUMBER->SKIP->=FailAndReload(Not a number)->
    COPY->GREATER(#25)->NOT->SKIP->=FailAndReload(Too big)->
    COPY->LESS(#0)->NOT->SKIP->=FailAndReload(What??)->
    MUL(#1000)->BURY->
    KEY(Infusionobs)->$[EpQ]->DIGUP->
    DOSQL(INSERT INTO INFUSIONOBS(infusionobs,Epoch,inoRate)VALUES
      ($[],$[],$[]))',

   'InfuRateSet');
```

We multiply by one thousand to convert from ml/hour to microlitres/hr.

What about actually setting up the infusion? *EpidInfuSet* does the job. Given the type of (for now, epidural) infusion, we check for an existing infusion. If one doesn't exist, we branch off to create the new infusion process and associate the drug with the infusion. We also create a new epoch on the infusion.

If the infusion exists, we make sure that the selected drug (mix) is the same. If it isn't, we confirm the change, branch off to stop the current infusion process, create a new process, and then the infusion and epoch.

If the infusion exists and the drug is the same, we check for a recent epoch on the process. If the epoch exists, we do nothing, as all is in order; otherwise we create the new epoch.

Because of the convenient pairwise structure of our lists, we provide the drug *ID* on the stack.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (190,
    '$[prQ]->SAME(#0)->NOT->SKIP->=NewRgnProcInfu->
    COPY->$[prQ]->
     QUERY(SELECT RX.Drug FROM RX WHERE
       RX.Process = $[])->
    SAME->SKIP->=ChangeEpidInfusion->
    $[EpQ]->SAME(#0)->SKIP->RETURN->
    $[prQ]->&NewEpoch->SET(EpQ)',

   'EpidInfuSet');
```

NewRgnProcInfu receives the drug ID on the stack. We create a new infusion process, attach to it an RX entry, and also make an epoch on the infusion (as above).

The following was epidural-specific but has now been tailored to make it generic for all PCA/infusions. This fix relies on the coding for regional procedures being from 100 to 160, the code for associated infusions being exactly 100 higher, and the code for *PCA* infusions 200 higher than the code for the regional procedure (i.e. 100 above the plain infusion code).

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (191,
    'BURY->
     $[prR]->QUERY(SELECT ProcType FROM PROCESS WHERE process = $[])->
     COPY->SAME(#109)->NOT->SKIP->ADD(#1)->
     ADD(#200)->
    CONFIRM(Using PCA?)->SKIP->SUB(#100)->&NewProc->SET(prQ)->
    KEY(Rx)->DIGUP->$[prQ]->
    DOSQL(INSERT INTO RX(rx,Drug,Process)
      VALUES($[],$[],$[]))->
    $[prQ]->&NewEpoch->MENU(0)',

   'NewRgnProcInfu');
```

The test for process type 109 is a search for a CSE process. If this is reported, we regard the infusion as a simple 'Epidural' and code accordingly (codes 210 and 310 for non-PCEA or PCEA infusions respectively).[66]

At the end of the routine we reload the menu rather than just saying SET(EpQ) because we then enable or disable the PCEA boxes as appropriate.[67]

The SKIP/SUB hack after the confirmation utilises our rather arbitrary convention that the ID infusion process with PCA is one hundred higher than the ID of the process without PCA — the process code for an epidural infusion is 210 and for an epidural *with PCEA* is 310.

Here is the simple *NewEpoch*, which might profitably be relocated elsewhere. The process ID must be submitted on the stack, and is consumed. The routine returns the new epoch ID on the stack.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (193,
    'BURY->KEY(Epoch)->COPY->NOW->ME->DIGUP->
    DOSQL(INSERT INTO EPOCH(epoch,oMade,Person,Process)
      VALUES($[],TIMESTAMP ''$[]'',$[],$[]))',

   'NewEpoch');
```

[66] The ADD(#1) is an ugly hack and the adventurous might replace this with the untested REPLACE(#110).

[67] Ultimately the best way of doing this would be by messages to the relevant boxes!

ChangeEpidInfusion receives the *new* drug ID on the stack. By our own convention, every time we change the drug being infused, we stop the current infusion process, and create an entirely new one. We do so, and then create a new infusion. Voilà!

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (192,
    'CONFIRM(Alter infusion?)->
    SKIP->=FailAndReload(Not changed!)->
    $[prQ]->&KillProc->&NewRgnProcInfu',

   'ChangeEpidInfusion');
```

GetInfusionLabel is simple, with one wrinkle — if there is no current epoch (EpQ) on the infusion process, it won't yet display the nature of the infusion. This is a 'safety' (checking) feature, preventing bias.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (250,
    '$[EpQ]->SAME(#0)->NOT->SKIP->RETURN(?)->
    $[prQ]->
    QUERY(SELECT DRUG.dTrade FROM RX,DRUG
      WHERE RX.Process = $[] AND
      RX.Drug = DRUG.drug)->QOK->SKIP->" "->COPY->
      IN(Fentanyl)->SET(fentanyl)',

   'GetInfusionLabel');
```

We look for the string "Fentanyl" within the value to be returned, and if this is present we set the fentanyl flag to #1.

TopupSet sets the number of topups, submitted on the stack. This can only be one number for each EPOCH epoch, so if a current RXOBS exists for this process, then we update it. We're looking at the epoch of the infusion process (EpQ), not the epidural one.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (196,
    'COPY->ISNUMBER->SKIP->=FailAndReload(Number please)->
     COPY->GREATER(#100)->NOT->SKIP->=FailAndReload(Too many?)->
     COPY->LESS(#0)->NOT->SKIP->=FailAndReload(What?)->
     $[EpQ]->SAME(#0)->
     NOT->SKIP->=FailAndReload(Please first state mix)->
     $[EpQ]->QUERY(SELECT RXOBS.rxobs FROM RXOBS WHERE
       RXOBS.Epoch = $[])->
     QOK->SKIP->&NewRxObs->
     DOSQL(UPDATE RXOBS SET rxoDoses=$[] WHERE rxobs = $[])',
 'TopupSet');
```

The subsidiary *NewRxObs* creates just that, returning the ID of the new RXOBS entry. The epoch `EpQ` is assumed to exist.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (198,
    'KEY(Rxobs)->COPY->$[EpQ]->
    DOSQL(INSERT INTO RXOBS(rxobs,Epoch)VALUES
      ($[],$[]))',
 'NewRxObs');
```

FindTopups locates an existing RXOBS on the current infusion process.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (197,
    '$[EpQ]->
     QUERY(SELECT RXOBS.rxoDoses FROM RXOBS
           WHERE RXOBS.Epoch = $[])',
 'FindTopups');
```

For PCEA, things are somewhat different from the above. On both getting/setting, we submit the epoch ID on the top of the stack, with the parameter to alter/find ID (or zero if nonexistent) below this.

In the case of *PcaRecord* below these two is the actual observed value — here we first test to make sure there *is* an epoch. We then validate the numeric epoch.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (199,
    'COPY->SAME(#0)->
      NOT->SKIP->=FailAndReload(Please state mix)->
    BURY->BURY->
    COPY->ISNUMBER->SKIP->=FailAndReload(A number please)->
    COPY->GREATER(#999)->
      NOT->SKIP->=FailAndReload(Too many)->
    COPY->LESS(#0)->NOT->SKIP->=FailAndReload(What?)->
    DIGUP->DIGUP->
    QUERY(SELECT PCA.pca FROM PCA WHERE PCA.Epoch = $[])->
    QOK->SKIP->&NewPca->
    BURY->SWOP->DIGUP->
    DOSQL(UPDATE PCA SET pco$[]=$[] WHERE PCA.pca = $[])',
 'PcaRecord');
```

The stack manipulation is fairly subtle. First we bury the epoch ID and nature of the update, so we might check the datum. If this check comes out ok, we then dig up the ID, and find the corresponding PCA table entry (or make one). Finally, we bury the PCA table entry, swop the datum type and value, and then dig up the entry id again!

The subsidiary *NewPca* resembles *NewRxObs*. We even (???) assume `EpQ` is filled. It returns the new ID value.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (201,
    'KEY(Pca)->COPY->$[EpQ]->
    DOSQL(INSERT INTO PCA(pca,Epoch)VALUES($[],$[]))',
 'NewPca');
```

GetPca is both ugly and trivial.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (200,
    'QUERY(SELECT PCA.pco$[] FROM PCA WHERE PCA.Epoch = $[])',
 'GetPca');
```

pcAble tests whether the submitted process (on the stack) is a PCA-associated process. If so, the item associated with the script is enabled, otherwise (by default) disabled.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (202,
    'QUERY(SELECT PROCESS.ProcType FROM PROCESS
                WHERE PROCESS.process = $[])->
    QOK->SKIP->#0->
    COPY->BURY->GREATER(#299)->DIGUP->
    LESS(#400)->AND->ENABLED',
 'pcAble');
```

If no process is found, we insert a zero, forcing 'disabling'. Our final contribution to the regional menu is a button to permit toggling between PCEA and no PCEA:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (425, 2,  'PCA is off',  'tglp', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (425, 904, 425, 35,
          0.60, 0.32, 0.35, 0.10, 0, 'yellow', 'black');
UPDATE ITEM SET iResponse = '&TogglePcea' WHERE iID = 425;
UPDATE ITEM SET iInitial =
       '&IsItPcra->SKIP->RETURN->"PCA is on"'
       WHERE iID = 425;
```

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (203,
    '$[prQ]->QUERY(SELECT PROCESS.ProcType FROM PROCESS
                WHERE PROCESS.process = $[])->
    QOK->SKIP->RETURN(#0)->
    COPY->GREATER(#299)->SWOP->LESS(#351)->AND',
 'IsItPcra');
```

The above was specific for PCEA but now covers all regional infusions (process type codes 300–350) so has been renamed from ‘IsItPcea’ to IsItPcra! QOK tests for success of the query, returning ‘no’ if no process found; the remaining code checks that the type is both over 299 AND under 351.

TogglePcea despite the label goes through the whole rigmarole of confirming a change and asking yes/no. The label on the button is simply a hint! We find the current drug and submit it to *NewRgnProcInfu*.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (204,
    '$[prQ]->QUERY(SELECT RX.Drug FROM RX WHERE RX.Process = $[])->
    QOK->SKIP->=Fail(Select Mix!)->
    &ChangeEpidInfusion',

 'TogglePcea');
```

The following ‘hypotension-related’ buttons are fleshed out in a later menu, but we define them here. Note the importance of having a distinct, identical miGroup value for the two buttons:

```
-- hypotension:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (704, 4,  'Y',  'bpY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (432, 904, 704, 20,
          0.45, 0.88, 0.07, 0.08, 9);
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (705, 4,  'N',  'bpN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (433, 904, 705, 21,
          0.55, 0.88, 0.07, 0.08, 9);
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (706, 1,  'low BP',  'bpL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (434, 904, 706, 22,
          0.45, 0.78, 0.26, 0.08, 0);
UPDATE ITEM SET iInitial =
       '#1120->&FetchProblem' WHERE iID = 704;
UPDATE ITEM SET iResponse =
       '#1120->&SpawnProblem->#1->&SetProblem'
       WHERE iID = 704;

UPDATE ITEM SET iInitial =
       '#1120->&FetchProblem->QOK->SKIP->#1->BOOLEAN->NOT'
```

```
       WHERE iID = 705;
UPDATE ITEM SET iResponse =
       '#1120->&SpawnProblem->#0->&SetProblem'
       WHERE iID = 705;
```

A final few buttons, to stop the infusion, and even remove the epidural (which can also be signalled by clicking 'N' in the preceding menu).

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (426, 2,  'End infusion',  'noinf', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (426, 904, 426, 37,
          0.03, 0.80, 0.35, 0.08, 0, 'yellow', 'black');
UPDATE ITEM SET iResponse = '&StopInfusion->MENU(0)' WHERE iID = 426;

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (427, 2,  'Regional out',  'epout', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (427, 904, 427, 38,
          0.03, 0.90, 0.35, 0.08, 0, 'yellow', 'black');
UPDATE ITEM SET iResponse =
       '#104->#151->&ConsiderStopping' WHERE iID = 427;

-- day count:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (431, 1,  'D ?',  'd', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (431, 904, 431, 20,
          0.76, 0.80, 0.10, 0.08, 0);
UPDATE ITEM SET iInitial = '$[prR]->&CountDays->"Day $[]"' WHERE iID = 431;
```

Here's the *CountDays* routine which, given a process, determines the integer number of days since the start of the process. Partial days aren't counted.

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(264,
  'QUERY(SELECT rStart FROM PROCESS WHERE process = $[])->
    TIMESTAMP->FLOAT->
    NOW->FLOAT->SWOP->SUB->
    INTEGER',
'CountDays');
```

Here's *StopInfusion*:

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (206,
    '$[prQ]->SAME(#0)->NOT->SKIP->=Fail(No infusion)->
    CONFIRM(End infusion?)->SKIP->=FailAndReload(Not stopped)->
    $[prQ]->&KillProc->ALERT(Stopped)',

 'StopInfusion');
```

5.10.3 Entering Details of the Regional Process

In the preceding text we've glossed over an important component: selecting the nature of the regional infusion. We've rather concentrated on epidurals, but we need to accommodate other regional infusions within our framework!

In the selection menu we must:

1. specify the type of infusion (e.g. epidural, sciatic nerve)
2. Confirm our choice, or cancel
3. Possibly specify other details of the infusion (later even have an optional comment; for now we limit things to further epidural details including catheter mark at skin in cm, and level of epidural).

Ultimately we should also make these data available on the actual regional menu.

[FIX THE FOLLOWING]

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (910, 20, 'Regional data', 'STARTREG');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (910, 910, 910, 0,
            0.001, 0.001, 0.990, 0.990, 0);
--- Self-reference.

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (1102, 6,  '',  'md',
  '->&GetRegionalModes');
UPDATE ITEM SET iResponse = 'SET(rmode)'
    WHERE iID = 1102;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1102, 910, 1102, 3,
          0.07, 0.20, 0.85, 0.08, 0);
```

```
INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (1103, 1,  'modality:',  'mdlbl');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1103, 910, 1103, 4,
          0.03, 0.05, 0.10, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (1104, 1,  'date in:',  'mdat');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1104, 910, 1104, 5,
          0.03, 0.40, 0.10, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (1105, 12,  '',  'date:');
-- type is 12 for date picker!
UPDATE ITEM SET iInitial =
  'NOW->SPLIT( )->DISCARD' WHERE iID = 1105;
UPDATE ITEM SET iResponse = 'SET(rdate)' WHERE iID = 1105;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1105, 910, 1105, 4,
          0.25, 0.40, 0.40, 0.08, 0);


INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1100, 2,  'Abort',  'abrt', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1100, 910, 1100, 10,
          0.05, 0.900, 0.200, 0.08, 0, 'green', 'white');
UPDATE ITEM SET iResponse = 'MENU(#1)'
  WHERE iID = 1100;
-- abort

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1101, 2,  'OK',  'ok', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1101, 910, 1101, 10,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
-- OK button
```

Let's initialise the menu:

```
UPDATE ITEM SET iInitial =
  'NAME(EpL)->#1->X->&LastEpoch->SET(EpL)->NAME(rmode)->
  NAME(rdate)->NOW->SPLIT( )->DISCARD->SET(rdate)->
  X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
  WHERE iID = 910;
```

Here's the response to clicking on the 'OK' button. We need to unload the current menu before we move to the regional one.

```
UPDATE ITEM SET iResponse =
 '$[rmode]->ISNULL->NOT->SKIP->=FailAndReload(Please select type of regional)->
  POPMENU(#0)->DISCARD->DISCARD->
  $[rmode]->SAME(#100)->NOT->SKIP->=NoteSpinal->
  $[rmode]->SAME(#101)->NOT->SKIP->=NoteSpinal->
  X->$[rmode]->$[rdate]->"$[] 00:00:00"->&DatedProc->
  MENU(REGIONAL)'
  WHERE iID = 1101;
```

Spinals are handled differently from all other processes. Whether the spinal is with or without morphine (codes 101 and 100 respectively) we make a note of the spinal and do *not* proceed to the regional menu.

At present we force the default time of insertion to 00:00:00, but we might of course insert a time field as well. The variable rmode contains the process type code for the regional. We use this to create a new process, and then enter the REGIONAL menu.

We need a routine to identify the various modes of regional infusion. Here they are (specified as *access* rather than infusion, which is separate).[68]

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (247,
    'QMANY(SELECT proctype,rptNature FROM PROCTYPE WHERE
     proctype > 99 AND proctype < 151)',
 'GetRegionalModes');
```

Code 109 is CSE, 110 is Epidural, and 120–150 are various regional catheter modalities.[69] Code 115 is a spinal infusion (sometimes used to good effect for palliative care patients). We now also include the special case of a single shot spinal (code 100) but need to handle this differently (Immediately close the process; have a check menu pop up after 24 hours).[70]

Here's the routine to record the single-shot spinal. We generate a process code, use X as the person, and fetch `rdate` as the date, turning it into a timestamp.

[68] We have established the convention that access+100 = infusion code!

[69] All rather arbitrary, and possibly better served by attaching a more generic process to an anatomical site?!

[70] To exclude the single shot spinal, use proctype $>$ 104 rather than 99.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (194,
    'KEY(Process)->X->$[rdate]->"$[] 00:00:00"->now->now->me->#100->
    DOSQL(INSERT INTO PROCESS
     (process,Person,rStart,rCreated,rEnd,rPlanner,ProcType)
     VALUES($[],$[],TIMESTAMP ''$[]'',TIMESTAMP ''$[]'',TIMESTAMP ''$[]'',$[],$[])
    X->#99->$[rdate]->"$[] 00:00:00"->&DatedProc->
    Alert(To check at +24hr!)->MENU(0)',
 'NoteSpinal');
```

We immediately terminate the process (spinal is code 100) by specifying rEnd.[71]

5.11 IV PCA menu (914)

Figure 11: IV PCA menu

The IV PCA menu has some resemblance to the Epidural menu, and is loosely based on this.

First we create the main menu. We use the same 'Done' button from the epidural menu, and also have a 'Remove PCA' button.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (914, 20, 'IV PCA', 'IVPCA');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH, miGroup)
               VALUES (914, 914, 914, 12,
                       0.000, 0.200, 0.999, 0.999, 0);
```

[71] A refinement might be to check whether the '24 hour check' is now due, and pop it up if this is the case! Easiest to modify Is24 appropriately, then test/skip using Is24.

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1401, 914, 401, 39,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1427, 2,  'Remove PCA',  'epout', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1427, 914, 1427, 40,
          0.03, 0.90, 0.35, 0.08, 0, 'yellow', 'black');
UPDATE ITEM SET iResponse =
       '#389->#391->&ConsiderStopping' WHERE iID = 1427;
```

We initialise as follows. The process code for IV PCA is 390. SetEvent(2) records our recent entry into the menu.

```
UPDATE ITEM SET iInitial =
  'NAME(EpL)->
   NAME(prQ)->NAME(EpQ)->
   NAME(fentanyl)->#0->SET(fentanyl)->
    #1->X->&LastEpoch->SET(EpL)->
    X->#389->#391->&FindAdministration->SET(prQ)->
    &SetEvent(#2)->
    &FindInfuObs->SET(EpQ)->
    X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
    WHERE iID = 914;
```

FindInfuObs relies on the presence of the variables prQ and EpL, which are the current infusion process and the most recent general observation, respectively. We need the NAME(fentanyl), for if this fentanyl variable is set to #1, then units are micrograms and not milligrams.

In the above we *assume* that prior to entry to the menu, if a PCA infusion doesn't exist, the user has confirmed the starting of such an infusion!

GoIvPca niggles if the user has already said 'no' (Hmm, get rid of this or modify it), then enters the menu if the process exists, otherwise confirms that the user wishes to make the process, and does so.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (207,
    'X->#389->#391->&FindAdministration->
    BOOLEAN->NOT->SKIP->MENU(IVPCA)->
    MENU(STARTIVPCA)',

 'GoIvPca');
```

Here are boxes related to PCA attempts and doses, as well as the total, and the popup for the ‘Mix’.

```
-- pcea good:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1421, 1,  'doses',  'gd', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1421, 914, 1421, 2,
          0.01, 0.21, 0.120, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1422, 14,  '',  'doseP', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1422, 914, 1422, 3,
          0.17, 0.21, 0.15, 0.08, 0);
UPDATE ITEM SET iResponse =
       'INTEGER->"Good"->$[EpQ]->&PcaRecord'
       WHERE iID = 1422;
UPDATE ITEM SET iInitial =
       '"Good"->$[EpQ]->&GetPca' WHERE iID = 1422;

-- pea tries:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1419, 1,  'tries',  'PtSrn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1419, 914, 1419, 4,
          0.34, 0.21, 0.120, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1420, 14,  '',  'tryT', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1420, 914, 1420, 5,
          0.49, 0.21, 0.15, 0.08, 0);
UPDATE ITEM SET iResponse =
       'INTEGER->"Tries"->$[EpQ]->&PcaRecord'
       WHERE iID = 1420;
UPDATE ITEM SET iInitial =
       '"Tries"->$[EpQ]->&GetPca' WHERE iID = 1420;
-- will always be enabled

-- total
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1417, 1,  'Total',  'tot', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
```

```
          miX, miY, miW, miH, miGroup)
    VALUES (1417, 914, 1417, 6,
          0.66, 0.21, 0.120, 0.08, 0);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1434, 1,  '(mg/mcg)',  'gd', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1434, 914, 1434, 7,
          0.66, 0.29, 0.120, 0.08, 0);


INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1418, 14,  '',  'totT', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1418, 914, 1418, 8,
          0.82, 0.21, 0.14, 0.08, 0);
UPDATE ITEM SET iResponse = '&ToMicrograms->INTEGER->&SetTotal' WHERE iID = 1418;
```

Here's the initialisation of the Total. We don't want to populate it with anything if FindTotal fails.

```
UPDATE ITEM SET iInitial =
  '$[EpQ]->&FindTotal->&FromMicrograms'
  WHERE iID = 1418;
```

We continue ...

```
-- basal infusion rate:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1415, 1,  'Basal: ML/h',  'PtSrn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1415, 914, 1415, 30,
          0.45, 0.70, 0.15, 0.08, 0);
-- QUERY: SHOULD THIS BE ML/HR OR MG/MCG PER HOUR?????????????
-- if latter need to alter the units dynamically!!

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1416, 14,  '',  'RateB', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1416, 914, 1416, 31,
          0.82, 0.70, 0.12, 0.08, 0);
UPDATE ITEM SET iResponse =
       'INTEGER->&InfuRateSet' WHERE iID = 1416;
UPDATE ITEM SET iInitial =
       '&FindInfuRate' WHERE iID = 1416;
```

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1423, 1,  'Mix',  '');
-- might even use epi 'Mix' item?

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1423, 914, 1423, 1,
          0.03, 0.001, 0.15, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (1424, 6,  '',  'pcai', '->#1->&ListDrugs');
-- code for PCA in DRUGUSAGE table is 1.
UPDATE ITEM SET iInitial =
       '&GetInfusionLabel' WHERE iID = 1424;
UPDATE ITEM SET iResponse =
       '&IvInfuSet' WHERE iID = 1424;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1424, 914, 1424, 2,
          0.16, 0.01, 0.82, 0.08, 0);


-- nausea (also used later in summary menu):
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (707, 4,  'Y',  'nvY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1425, 914, 707, 26,
          0.22, 0.69, 0.07, 0.08, 3);
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (708, 4,  'N',  'nvN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1426, 914, 708, 27,
          0.31, 0.69, 0.07, 0.08, 3);
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (709, 1,  'nausea',  'nvL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1429, 914, 709, 28,
          0.03, 0.69, 0.12, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (716, 2,  'nausea Rx',  'nvL');
UPDATE ITEM SET iResponse =
```

```
  'MENU(NAUSEA)' WHERE iID = 716;

-- do NOT enable the following [find the bug first!]
--INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
--          miX, miY, miW, miH, miGroup)
--  VALUES (1428, 914, 716, 3,
--          0.05, 0.75, 0.33, 0.08, 0);

UPDATE ITEM SET iInitial =
       '#1130->&FetchProblem' WHERE iID = 707;
UPDATE ITEM SET iResponse =
       '#1130->&SpawnProblem->#1->&SetProblem'
       WHERE iID = 707;

UPDATE ITEM SET iInitial =
       '#1130->&FetchProblem->QOK->SKIP->#1->BOOLEAN->NOT'
       WHERE iID = 708;
UPDATE ITEM SET iResponse =
       '#1130->&SpawnProblem->#0->&SetProblem'
       WHERE iID = 708;

--- ========================finally, bolus and lockout values:

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1431, 1,  'Bolus',  'gd', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1431, 914, 1431, 15,
          0.01, 0.45, 0.120, 0.08, 0);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1433, 1,  '(mg/mcg)',  'gd', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1433, 914, 1433, 16,
          0.01, 0.54, 0.120, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1432, 14,  '',  'bolP', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1432, 914, 1432, 17,
          0.17, 0.45, 0.20, 0.08, 0);
```

Here's how we record PCA Dose settings.

```
UPDATE ITEM SET iResponse =
       '&ToMicrograms->INTEGER->
       "Dose"->$[EpQ]->&PcaNoteSettings'
       WHERE iID = 1432;
```

Because we variably enter doses in micrograms (for fentanyl) or milligrams (for everything else) but always record *micrograms*, we need to be able to convert to micrograms (where needed) prior to storing a number. Here's the routine, which is complicated by the possibility that a value of eg 0.5 has been entered. So we need to convert to a float, multiply, and then convert back to an integer.[72] This routine assumes that the variable $[fentanyl] exists, and that a number is supplied on the stack. We return a float, which must subsequently be converted to an integer. (In the case of fentanyl we return the type which was input).

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (273,
    '$[fentanyl]->NOT->SKIP->RETURN->
    FLOAT->FLOAT(1000)->MUL',

   'ToMicrograms');
```

Here's the converse routine. The shenanigans at the end are because on the PDA 0.5 is unacceptably rendered "5.000000e-1".

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (274,
    'COPY->ISNULL->NOT->SKIP->RETURN->
    $[fentanyl]->NOT->SKIP->RETURN->
    FLOAT->FLOAT(1000)->DIV->COPY->
    FLOAT(1.0)->LESS->NOT->SKIP->=TrimFloat->
    INTEGER',

   'FromMicrograms');
```

We return NULL if the submitted value is NULL. If the value is 1 or more, then we turn it into an integer.[73] If the value is under 1.0, then we trim the float as follows:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (275,
    'FLOAT(10)->MUL->INTEGER->"0.$[]"',
   'TrimFloat');
```

The above will never be applied to fentanyl dosing, but e.g. 5.000000000e-1 will become 0.5.

Here's the initialisation of the PCA bolus dose setting where *FromMicrograms* is used. We perform the reverse of the preceding, dividing by 1000 *unless* we're dealing with fentanyl! [NOTE THAT AT PRESENT on the PDA we are reduced to blasted scientific notation. Need to convert 5.00000e-1 to 0.5 !!]

[72] Once we've got our fixed point routines working, things will be considerably easier!

[73] We truncate e.g. 1.5 mg. This might withstand a gentle rewrite!

```
UPDATE ITEM SET iInitial =
 '"Dose"->$[EpQ]->&GetPcaSet->
  &FromMicrograms' WHERE iID = 1432;
```

We continue ...

```
-- pca tries:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1439, 1,  'lockout (min)',  'lkL', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1439, 914, 1439, 18,
          0.45, 0.45, 0.120, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1440, 14,  '',  'lkT', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1440, 914, 1440, 19,
          0.82, 0.45, 0.12, 0.08, 0);
UPDATE ITEM SET iResponse =
       'FLOAT->FLOAT(60)->MUL->INTEGER->"Lockout"->$[EpQ]->&PcaNoteSettings'
       WHERE iID = 1440;
UPDATE ITEM SET iInitial =
       '"Lockout"->$[EpQ]->&GetPcaSet->COPY->ISNULL->SKIP->DIV(#60)'
       WHERE iID = 1440;
```

We multiply the value by 60 for the Lockout setting above, as lockout is recorded in seconds but entered in minutes. The reverse must be performed on retrieving the value.

IvInfuSet is distressingly similar to *EpidInfuSet*, and ultimately we might aim for a common routine. The difference is that with the latter, we have no certainty that an infusion already exists but with this routine we have already confirmed the presence of PCA. We still may not know the type of infusion, however! We have the ID of the drug on the stack.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (208,
    'COPY->
     COPY->SAME(#122)->SET(fentanyl)->
     $[prQ]->
     QUERY(SELECT RX.Drug FROM RX WHERE
       RX.Process = $[])->
     QOK->SKIP->=NewIvDrug->
    SAME->SKIP->=ChangeIvInfusion->
    DISCARD->
    $[EpQ]->SAME(#0)->SKIP->RETURN->
```

```
    $[prQ]->&NewEpoch->SET(EpQ)',

  'IvInfuSet');
```

First of all we find the ID of the drug, and make a copy of this ID. The SAME(#122) is something of a hack, as this is the code for fentanyl.[74]

We then test whether a drug is attached to the IV PCA process. If not, we branch off and attach the drug using *NewIvDrug*.[75]

Otherwise, we compare the attached drug with the copy of our current drug. If they're not the same, we alter the infusion (*ChangeIvInfusion*, using the original copy of the drug ID still on the stack.

Lastly we ensure that the currrent epoch (EpQ) on the infusion process exists, for if it doesn't we have to create a new epoch on that process!

NewIvDrug accepts a drug ID on the stack, and creates an RX entry which refers to the process prQ. It also attaches a new epoch to the process.

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (209,
    'BURY->KEY(Rx)->DIGUP->$[prQ]->
    DOSQL(INSERT INTO RX(rx,Drug,Process)
      VALUES($[],$[],$[]))->
    $[prQ]->&NewEpoch->SET(EpQ)',

    'NewIvDrug');
```

Here's *ChangeIvInfusion* followed by *NewIvInfuProc*. The former simply confirms the change, the latter is more involved.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (210,
    'CONFIRM(Alter infusion?)->
    SKIP->=FailAndReload(Not changed!)->
    $[prQ]->&KillProc->&NewIvInfuProc->
    MENU(#0)',

   'ChangeIvInfusion');
```

NewIvInfuProc accepts the ID of the drug on the stack. A new process is created, followed by a call to *NewIvDrug*. The following is all related specifically to PCA, rather than any old IV infusion.

hypertargetNewIvInfuProc

[74] Such hard-coding is a really bad idea!

[75] A spare copy of the drug ID is left on the stack afterwards. This is ugly.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (211,
    '&NewProc(#390)->SET(prQ)->
    &NewIvDrug',

   'NewIvInfuProc');
```

The code for IV PCA is 390.

5.11.1 Noting the PCA settings

Similar to *PcaRecord* is *PcaNoteSettings*[76] At present we only write to the psoDose and psoLockout fields in the PCASETTINGS table. [We need more diligent checks on the values input — FIX ME, even have separate table to describe these].

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (212,
    'COPY->SAME(#0)->NOT->SKIP->=FailAndReload(Please state mix)->
    BURY->BURY->
    COPY->ISNUMBER->SKIP->=FailAndReload(A number please)->
    DIGUP->DIGUP->
    QUERY(SELECT PCASETTINGS.pcasettings FROM PCASETTINGS
          WHERE PCASETTINGS.Epoch = $[])->
    QOK->SKIP->&NewPcaSet->
    BURY->SWOP->DIGUP->
    DOSQL(UPDATE PCASETTINGS SET pse$[]=$[]
          WHERE PCASETTINGS.pcasettings = $[])',
 'PcaNoteSettings');
```

[WE MUST FIX THE ABOVE. pseDose should be in micrograms, pseLockout in seconds!]

NewPcaSet resembles *NewPca*.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (213,
    'KEY(Pcasettings)->COPY->$[EpQ]->
    DOSQL(INSERT INTO PCASETTINGS(pcasettings,Epoch)VALUES($[],$[]))',
 'NewPcaSet');
```

Likewise *GetPca* and *GetPcaSet*.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (214,
    'QUERY(SELECT PCASETTINGS.pse$[] FROM PCASETTINGS
           WHERE PCASETTINGS.Epoch = $[])->QOK->SKIP->NULL',
 'GetPcaSet');
```

[76] One is almost tempted to merge the the PCA and PCASETTINGS tables into one.

For totals, the eponymous *FindTotal*. This accepts the relevant epoch to which the RXOBS refers:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (215,
    'QUERY(SELECT RXOBS.rxoTotal FROM RXOBS
      WHERE RXOBS.Epoch = $[])->QOK->SKIP->NULL',

   'FindTotal');
```

We return NULL if the query failed.

SetTotal is a little more convoluted, and assumes that EpQ is the epoch. The value is on the stack. If an RXOBS entry doesn't exist and the SQL therefore fails, we create an entry.[77]

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (216,
    '$[EpQ]->SAME(#0)->NOT->SKIP->=Fail(Select Mix)->
     COPY->ISNUMBER->SKIP->=Fail(Number please)->
     $[EpQ]->QUERY(SELECT rxobs FROM RXOBS WHERE Epoch = $[])->
     QOK->SKIP->=SetNewTotal->DISCARD->
     $[EpQ]->
     DOSQL(UPDATE RXOBS SET rxoTotal=$[] WHERE
       Epoch = $[])->
     QOK->SKIP->&Fail(Failed to set total)->RETURN',

   'SetTotal');
```

Given the new value on the top of the stack and the epoch ID in EpQ, insert a new total Rx value using *SetNewTotal*:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (251,
    'BURY->KEY(Rxobs)->$[EpQ]->DIGUP->
     DOSQL(INSERT INTO RXOBS(rxobs,Epoch,rxoTotal)VALUES
       ($[],$[],$[]))',

   'SetNewTotal');
```

5.12 Start menu for PCA (913)

It's desirable to be able to specify a different start date from 'now', and this is the sole justification for the menu:

[77] A clumsiness is the value left on the stack if the entry does exist!

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (913, 20, 'Start IV PCA', 'STARTIVPCA');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (913, 913, 913, 0,
            0.001, 0.001, 0.990, 0.990, 0);
--- Self-reference.

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1302, 1, 'Start IV PCA?', 'l');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (1302, 913, 1302, 0,
            0.25, 0.15, 0.50, 0.08, 0);

-- 1104 is 'date in':
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1304, 913, 1104, 5,
          0.03, 0.42, 0.10, 0.08, 0);

-- 1105 is a date picker, as for epidural:
-- we use the $[rdate] variable.
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1305, 913, 1105, 4,
          0.25, 0.40, 0.40, 0.08, 0);

-- 1100 is an 'Abort' button:
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1300, 913, 1100, 10,
          0.05, 0.900, 0.200, 0.08, 0, 'green', 'white');

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1301, 2,  'OK',  'ok', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1301, 913, 1301, 10,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
-- OK button
```

Let's initialise the menu:

```
UPDATE ITEM SET iInitial =
  'NAME(rdate)->NOW->SPLIT( )->DISCARD->SET(rdate)->
```

```
  X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
  WHERE iID = 913;
```

Here's the response to clicking on the 'OK' button. We need to unload the current menu before we move to the regional one.

```
UPDATE ITEM SET iResponse =
 'POPMENU(#0)->DISCARD->DISCARD->
  X->#390->$[rdate]->"$[] 00:00:00"->&DatedProc->
  MENU(IVPCA)'
  WHERE iID = 1301;
```

5.13 Oral therapy menu (915)

Technically this is *enteral* rather than oral therapy, as we will also cover aspects such as nasogastric and nasojejunal administration of drugs. At present we cover these all under the rubric of 'oral' (process code 50) but this approach would benefit from revision.

Figure 12: Oral therapy

Let's create a menu with a 'Done' button, and a 'Stop ALL' button:

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (915, 20, 'Orals', 'ORALS');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH, miGroup)
               VALUES (915, 915, 915, 12,
                       0.000, 0.001, 0.999, 0.999, 0);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
```

```
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1501, 915, 401, 2,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1502, 2,  'Stop ALL orals',  'epout', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1502, 915, 1502, 2,
          0.03, 0.90, 0.48, 0.08, 0, 'yellow', 'black');
UPDATE ITEM SET iResponse =
       '#49->#51->&ConsiderStopping' WHERE iID = 1502;
```

We initialise by determining the most recent general observation for this patient, that is we find the start of the current epoch. SetEvent(3) records our recent entry into the Orals menu.

```
UPDATE ITEM SET iInitial =
  'NAME(EpL)->
   #1->X->&LastEpoch->SET(EpL)->&SetEvent(#3)->
   X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
    WHERE iID = 915;
```

We also need to be able to enter the ORALS menu. Here's the routine. Remember that oral therapy is process code 50 (between 49 and 51).

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (218,
    'X->#49->#51->&FindAdministration->
    BOOLEAN->NOT->SKIP->MENU(ORALS)->
    CONFIRM(On orals?)->SKIP->=FailAndReload(No!)->
    MENU(ORALS)',

 'GoOrals');
```

Next we need to insert controls to process the following:

1. A *list* of current oral drugs, each with an associated tick box (KISS)
2. Ability to remove a drug from the list by simply clicking on the name
3. Ability to add another drug by selecting a drug from a list

5.13.1 A list of oral drugs

```
INSERT INTO ITEM (iID, iType, iText, iName, iLines)
  VALUES (1508,8,'[No current Rx]','PoRx',8);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1512,  2,     '',    'Nam' ),
                 (1510,  14,     '',    'tot' );

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (1508, 915, 1508, 0, 0.001, 0.12, 0.95, 0.74);

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
                       irName, irFraction, irEnabled)
  VALUES (1512, 1508, 1512, 1, 'Drug',      0.70, 1),
         (1510, 1508, 1510, 2, 'mg in 24h',        0.29, 1);

UPDATE ITEM SET iInitial = 'X->#49->#51->&GetDrugProcs' WHERE iID = 1508;
```

Here's the *GetDrugProcs* routine. Note the similarity to *ProcBetween* but this routine returns a list rather than the first hit!

```
INSERT INTO FUN (fKey, fBody, fName) VALUES (220,

  'QMANY(SELECT PROCESS.process FROM PROCESS WHERE
       PROCESS.rEnd IS NULL AND
       PROCESS.Person = $[] AND
       PROCESS.ProcType > $[] AND
       PROCESS.ProcType < $[])',

'GetDrugProcs');
```

We also need to initialise the components:

```
UPDATE ITEM SET iInitial = 'V->&GetTradeName'
  WHERE iID = 1512;
```

GetTradeName allows us to determine the trade name, given the process to which the drug is attached.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (231,
    'QUERY(SELECT DRUG.dTrade FROM RX,DRUG
  WHERE RX.Process = $[] AND
    RX.Drug = DRUG.drug)',
 'GetTradeName');
```

And respond to a click on the name button by giving the opportunity to stop the drug:

```
UPDATE ITEM SET iResponse = 'V->&AskStopDrug'
  WHERE iID = 1512;
```

AskStopDrug is a response to clicking on a drug-named button.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (232,
    'QUERY(SELECT DRUG.dTrade FROM RX,DRUG
  WHERE RX.Process = $[] AND
    RX.Drug = DRUG.drug)->
  CONFIRM(Stop the $[]?)->
  SKIP->RETURN->
  NOW->V->DOSQL(UPDATE PROCESS SET rEnd=TIMESTAMP ''$[]''
          WHERE PROCESS.process = $[])->
  MENU(0)',
 'AskStopDrug');
```

We don't fill in any values, but the existence of the epoch tells us that therapy was given today.[78]

Here's *NewRxObs2*, which should be renamed. It accepts a process ID on the stack and returns the corresponding RxObs ID, or if none exists, makes one and returns it.

```
INSERT INTO FUN (fKey, fBody, fName) VALUES (223,

  'COPY->$[EpL]->
  QMANY(SELECT MAX(EPOCH.epoch) FROM EPOCH WHERE EPOCH.Process = $[]
    AND EPOCH.epoch > $[])->
  QOK->SKIP->&FancyEpoch->
  COPY->QUERY(SELECT RXOBS.rxobs FROM RXOBS
    WHERE RXOBS.Epoch = $[])->
  QOK->NOT->SKIP->RETURN->
  BURY->KEY(Rxobs)->COPY->DIGUP->
  DOSQL(INSERT INTO RXOBS(rxobs,Epoch)
    VALUES($[],$[]))',

'NewRxObs2');
```

This routine is nasty as it returns a ragged stack — if there is already an RXOBS then the copy of the EPOCH is left below this. Hmm.

[78] Hmm. what if we cancel this??

We pull out the trade name of the drug,[79] and only tick the box if an epoch has been made on the process more recently than the generic observation for the whole process.

We must also record and reveal the total amount of drug given, if noted. We attach this functionality to the numeric box:

```
UPDATE ITEM SET iResponse = '&Set24'
  WHERE iID = 1510;
```

Set24 sets the most recent (24 hour) dosage.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (233,
    'INTEGER->COPY->ISNUMBER->SKIP->=Fail(Not a number!)->
   BURY->V->&NewRxObs2->
   DIGUP->SWOP->
   DOSQL(UPDATE RXOBS SET rxoTotal=$[] WHERE rxobs = $[])',
 'Set24');
```

In the above we first 'convert to an integer' (from a text string) and then use the cumbersome 'isnumber' test.[80]

```
UPDATE ITEM SET iInitial = 'V->&Get24hr'
  WHERE iID = 1510;
```

Get24hr obtains the most recently recorded (24 hour) dosage.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (234,
    '$[EpL]->
  QUERY(SELECT RXOBS.rxoTotal
  FROM RXOBS,EPOCH WHERE RXOBS.Epoch = EPOCH.epoch AND
  EPOCH.Process = $[] AND EPOCH.epoch > $[])',
 'Get24hr');
```

Here are the buttons controlling the addition of another drug. Process code 50 is 'enteral drug administration'. We include access to our 'nausea Rx menu' both here and later!

[79] A later option might be the ability to check the generic name using a stylus tap!

[80] It's most desirable to have INTEGER perform the test, and then be able to test for/interrupt on this condition. A lot of overhead could be removed if we instituted such 'parallel' evaluation!

5.13.2 ListDrugs

It was good to take out the PHARM table as it isn't needed on the PDA (but might later be introduced for analysis etc). We have now created a DRUGUSAGE table, which associates a DRUG entry with a particular role. Important roles are PCA and anti-nauseant. For details, see *AnalgesiaDBpart1.tex*.

We can now identify the formulation of the drug (for e.g. epidural use, IV use, TTS patch, or whatever) AND the specific role(s) of each formulation of each drug. We might make the DRUGUSAGE table sparse, so that if we don't require a role, then we don't busy up this table!

We don't need a separate epidural role for formulations which are 'epidural'; the fun arises with PCA drugs (which can otherwise be used IV), the variety of anti-nauseants, which can have various formulations, and oral analgesics.

All instances of use of ListDrugs will then be modified to remove the (eugh) dependence on particular key codes (in a range), replacing such usage with the more robust *role identification*.

```
INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (1517, 6,  'ADD Analgesic',  'newdrg',
  '->#5->&ListDrugs');
-- Usage value of 5 in DRUGUSAGE = oral analgesics

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1716, 915, 716, 3,
          0.68, 0.001, 0.30, 0.08, 0);

UPDATE ITEM SET iResponse = '#50->&SetDrug' WHERE iID = 1517;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (1517, 915, 1517, 2, 0.001, 0.001, 0.60, 0.10);
```

ListDrugs at present simply lists all options.[81] As usual in a list, we provide pairs of IDs and items. *ListDrugs* previously looked for primary keys in a range. It now consults the DRUGUSAGE table for the relevant code, joining the resulting DRUG table keys.

```
INSERT INTO FUN (fKey, fBody, fName) VALUES (221,

  'QMANY(SELECT DRUGUSAGE.Drug,DRUG.dTrade FROM DRUGUSAGE,DRUG WHERE
    DRUGUSAGE.drUsage = $[] AND DRUGUSAGE.Drug = DRUG.drug)',
```

[81] Might refine to exclude items already selected!

```
'ListDrugs');
```

Formerly we listed drugs with a reference to a 'pharmaceutical agent' table (PHARM), using the primary key of that table to specify a range within which items were selected. This approach was nasty for two reasons:

1. Compound drugs (mixtures containing several active ingredients) are not well represented;

2. The use of a range of keys is convenient but artificial and ugly.

We have re-written things to permit grouping of drugs in several ways. We retain grouping by ID (but this time, for the drug to be selected, the ID must be within a range of key numbers specified in the DRUG table!), and also permit grouping by formulation. This approach frees up the PHARM table allowing us (if we wish) to associate several drugs with one particular formulation, but still maintains the rather unfortunate key-based selection, simply because it's so fast and convenient. As things stand, we have completely removed the PHARM table, but it can be re-introduced and then associated in a many to one fashion with items in the DRUG table using, say, a PHARMDRUG table. This latter approach would permit us to represent compound drugs, if we wanted to.

Invocations of ListDrugs were as follows:

Item 424 Regional infusion; (was: 99–110) drugform: 1=epidural

Item 1424 IV infusion; (was: 119–130) drugform: 8 = for intravenous PCA

Item 1517 Add oral analgesic; (was: 199–600) ((drugform: 10—13—17 = oral (various)))

Item 2517 PR drug administration; (was: 1999–2100) drugform: 20 = suppository

Item 2617 Transdermal drug; (was: 2999–3100) drugform: 30 = patch

Item 2717 Special infusion; (was: 699–899) drugform: 4 (IV non-pca)

Item 3517 Anti-nauseants; (was: 599-700) no particular drugform.

All of the above classes can actually be identified by formulation, apart from the anti-nauseants, oral analgesics and special infusions. We will rather arbitrarily allocate anti-nauseants DRUG key codes between 19456 and 20479 inclusive, allowing us to pull these out! Perhaps less nasty ways of classifying such drugs

would be (a) to describe classes in an associated table (slower) or (b) to have a classification field in the DRUG table, preferably using bit flags rather than constraining each agent to a single class (complex and requires bit-masking in SQL = ugh).

We will similarly allocate oral analgesics codes between 2048 and 5119 inclusive, and 'special infusions' even more arbitrary codes between 22528 and 100351.

Here's a routine to list by formulation (given the formulation code), for example 1 for regional, 2 for IV PCA formulations, 20 for rectal.

```
INSERT INTO FUN (fKey, fBody, fName) VALUES (265,

  'QMANY(SELECT drug,dTrade FROM DRUG WHERE
    DRUG.DrugForm = $[])',

'ByFormulation');
```

SetDrug creates a new process associated with administration of the selected drug, and then reloads so that the drug is now listed. [Think about having confirmation??] On the top of the stack is the nature of the drug therapy, and below this is the ID of the drug itself.

```
INSERT INTO FUN (fKey, fBody, fName) VALUES (222,

  'SWOP->COPY->BURY->
   QUERY(SELECT dTrade FROM DRUG WHERE drug = $[])->
   CONFIRM(Start $[]?)->SKIP->RETURN->
   &NewProc->
   BURY->KEY(Rx)->DIGUP->DIGUP->
   DOSQL(INSERT INTO RX(rx,Process,Drug)
      VALUES($[],$[],$[]))->
   MENU(0)',

'SetDrug');
```

5.14 Nausea Rx (3915)

This menu is almost a clone of the 'Oral therapy menu', despite the fact that anti-nauseants in this menu can be either oral or parenteral. We thought about this one a lot, and felt that bunging anti-nauseants on the pain page was more clumsy than putting them on the following page where nausea is listed as a problem.[82]

A menu with a 'Done' button:

[82] A new user will only fret once about the absence of these drugs on the pain page, and the nausea menu is also accessible from the 'orals' and 'other Rx' menus!

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (3915, 20, 'Nausea Rx', 'NAUSEA');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH, miGroup)
               VALUES (3915, 3915, 3915, 12,
                       0.000, 0.001, 0.999, 0.999, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (501, 2,  'Done',  'Exitbtn', '',  1);
UPDATE ITEM SET iResponse = 'MENU(-1)'
  WHERE iID = 501;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (3501, 3915, 501, 2,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
```

Button 501 is a similar to 401, a previously defined exit button. We initialise by determining the most recent general epoch for this patient, that is we find the start of the current epoch.

```
UPDATE ITEM SET iInitial =
  'NAME(EpL)->
   #1->X->&LastEpoch->SET(EpL)->
   X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
    WHERE iID = 3915;
```

Next we need to insert controls to process the following:

1. A *list* of current anti-nausea drugs
2. Ability to remove a drug from the list by simply clicking on the name
3. Ability to add another drug by selecting a drug from a list

5.14.1 A list of antinauseants

```
INSERT INTO ITEM (iID, iType, iText, iName, iLines)
  VALUES (3508,8,'[No nausea Rx]','nRx',8);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (3512,  2,     '',    'nNa' ),
                 (3510,  14,     '',    'nto' );

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (3508, 3915, 3508, 0, 0.001, 0.12, 0.95, 0.74);
```

```
INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
                       irName, irFraction, irEnabled)
  VALUES (3512, 3508, 3512, 1, 'Drug',      0.70, 1),
         (3510, 3508, 3510, 2, 'mg in 24h',        0.29, 1);

UPDATE ITEM SET iInitial = '&GetNauseaRxProcs' WHERE iID = 3508;
```

The following is a very specific routine to obtain all anti-nausea Rx for this patient alone:

```
INSERT INTO FUN (fKey, fBody, fName) VALUES (254,

  'X->QMANY(SELECT PROCESS.process FROM RX,PROCESS WHERE
       RX.Process = PROCESS.process AND
       RX.Drug > 19000 AND
       PROCESS.rEnd IS NULL AND
       PROCESS.Person = $[])',
'GetNauseaRxProcs');
```

At present we *still* use the clumsy hack of having key values for anti-nauseants of over 19000, rather than trying the more complex multiple join on the DRUGUSAGE table!

The Drug conditions in the above are outrageous hacks and really should be addressed! There is a further problem in the Ocelot database, where if we add in the condition 'RX.Drug $<$ 4700 AND' then we obtain an obscure GPF.[83]

We also need to initialise the components:

```
UPDATE ITEM SET iInitial = 'V->&GetTradeName'
  WHERE iID = 3512;
```

Respond to a click on the name button by giving the opportunity to stop the drug:

```
UPDATE ITEM SET iResponse = 'V->&AskStopDrug'
  WHERE iID = 3512;
```

We must also record and reveal the total amount of drug given, if noted. We attach this functionality to the numeric box:

```
UPDATE ITEM SET iResponse = '&Set24'
  WHERE iID = 3510;

UPDATE ITEM SET iInitial = 'V->&Get24hr'
  WHERE iID = 3510;
```

[83] Look into this; at some later stage 'higher' drugs will otherwise start appearing in this menu!

```
INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (3517, 6,  'ADD Nausea Rx',  'newdrg',
  '->#4->&ListDrugs');
-- 4 represents 'anti-nauseants' in the DRUGUSAGE table.

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (3517, 3915, 3517, 2, 0.001, 0.001, 0.60, 0.10);
```

ListDrugs looks at generic codes. The response to ADDing a drug is a lot more complex than is the case for orals, because we might be adding an IV drug, oral agent or even a scopolamine TTS patch! Our coding is clumsy, and depends heavily on the drug codes (See *PerlPgm.tex*, section on 'drug.csv'). This unpleasant coding subclassifies IV anti-nauseants into the range 19456–19967, orals from 19968–20096, and (at present) the remaining higher codes for 'transdermal'.[84] Process code 50 is enteral drug; 291 is IV drug boluses, 280 is transdermal.

```
UPDATE ITEM SET iResponse =
 'COPY->GREATER(#19967)->SKIP->=SetDrug(#291)->
  COPY->GREATER(#20096)->SKIP->=SetDrug(#50)->&SetDrug(#280)'
WHERE iID = 3517;
```

5.15 Other drugs and modalities (2915)

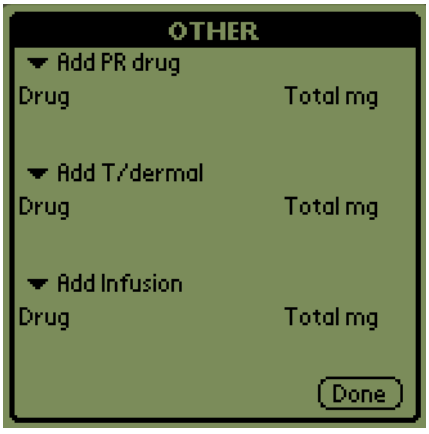


Figure 13: Other modalities

Set this up to accommodate:

1. Rectal paracetamol

[84] A really nasty hack. Ugh.

2. Rectal morphine
3. Rectal diclofenac
4. Transdermal clonidine
5. Transdermal fentanyl
6. SC morphine
7. IV ketamine infusions
8. Boluses of analgesics and other drugs.

First, we create the menu, with a ‘Done’ button…

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2915, 20, 'Other Rx', 'OTHER');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH, miGroup)
               VALUES (2915, 2915, 2915, 12,
                       0.000, 0.001, 0.999, 0.999, 0);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (2501, 2915, 401, 2,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
```

As for orals, we set EpL:

```
UPDATE ITEM SET iInitial =
  'NAME(EpL)->
   #1->X->&LastEpoch->SET(EpL)->&SetEvent(#4)->
   X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
    WHERE iID = 2915;
```

Here’s the entry routine. Process codes 260–299 represent the relevant ones!

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (229,
    'X->#259->#300->&FindAdministration->
    BOOLEAN->NOT->SKIP->MENU(OTHER)->
    CONFIRM(Other Rx?)->SKIP->=FailAndReload(No!)->
    MENU(OTHER)',

 'GoOther');
```

5.15.1 Rectal (PR) therapy

We create a table of ‘other Rx’:

```
INSERT INTO ITEM (iID, iType, iText, iName, iLines)
  VALUES (2508,8,'[No other Rx]','oRx',8);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2512,  2,     '',    'Nam' ),
                 (2510,  14,     '',    'tot' );

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (2508, 2915, 2508, 0,
                       0.001, 0.18, 0.95, 0.70);

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
                       irName, irFraction, irEnabled)
  VALUES (2512, 2508, 2512, 1, 'Drug',      0.70, 1),
         (2510, 2508, 2510, 2, 'mg in 24h',        0.29, 1);

UPDATE ITEM SET iInitial = 'V->&Get24hr'
   WHERE iID = 2510;

UPDATE ITEM SET iResponse = '&Set24'
  WHERE iID = 2510;

UPDATE ITEM SET iInitial = 'X->#259->#300->&GetDrugProcs'
   WHERE iID = 2508;

UPDATE ITEM SET iResponse = 'V->&AskStopDrug'
  WHERE iID = 2512;

UPDATE ITEM SET iInitial = 'V->&GetTradeName'
  WHERE iID = 2512;
```

We have buttons to add the various modalities:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (2517, 6,  'Add PR drug',  'newdrg',
  '->#20->&ByFormulation');

UPDATE ITEM SET iResponse = '#270->&SetDrug' WHERE iID = 2517;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (2517, 2915, 2517, 2,
```

```
          0.001, 0.001, 0.49, 0.08);

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (2617, 6,  'Add T/dermal',  'newdrg',
  '->#30->&ByFormulation');

UPDATE ITEM SET iResponse = '#280->&SetDrug' WHERE iID = 2617;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (2617, 2915, 2617, 3,
         0.501, 0.001, 0.49, 0.08);


INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (2717, 6,  'Add Infusion',  'newdrg',
  '->#3->&ListDrugs');
-- in the DRUGUSAGE table, 3 represents a 'special' IV infusion.

UPDATE ITEM SET iResponse = '#290->&SetDrug' WHERE iID = 2717;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (2717, 2915, 2717, 4,
          0.001, 0.10, 0.49, 0.08);

--- finally, IV boluses: (any IV with formulation code 4)
INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (2718, 6,  'IV boluses',  'newdrg',
  '->#2->&ListDrugs');
-- This gives IV drugs administered as IV boluses (excluding anti-nauseants)!

UPDATE ITEM SET iResponse = '#291->&SetDrug' WHERE iID = 2718;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (2718, 2915, 2718, 5,
          0.501, 0.10, 0.49, 0.08);
```

5.15.2 Yet more therapy

There are infinite possibilities. Best here might be to simply add a comment (Hmm) rather than having an even more busy menu!

5.16 ‘Finally’: New alerts &c (909)

Figure 14: New alerts

We create an ‘Alerts’ menu:

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (909, 20, 'New Alerts', 'FINISH');
```

Here’s the initialisation string:

```
UPDATE ITEM SET iInitial = 'NAME(EpL)->
   #1->X->&LastEpoch->SET(EpL)->
   X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
   WHERE iID = 909;
```

As usual we make the menu item self-referential, and create a few buttons at the bottom. We allow comment insertion here, again, and also have a ‘Discharge’ button at the bottom. The Done button must turn off CACHING by saying UNCACHE(PROCESS) ....

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem,
miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (909, 909, 909, 0,
            0.001, 0.001, 0.990, 0.990, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (700, 2,  'Done',  'exAl', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (700, 909, 700, 41,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
```

Here's the response to the [Done] button:

```
UPDATE ITEM SET iResponse =
   'X->&UnFlagMe->
   $[EpL]->&EndEpoch->
   UNCACHE(EPOCH)->UNCACHE(PROCESS)->MENU(4)'
  WHERE iID = 700;
```

UnFlagMe clears the '!' i.e. 'patient to be seen' flag in the list of patients on the ward, and EndEpoch records the time spent seeing this patient (in seconds) for the epoch stored in **EpL**. The MENU(4) command removes all stacked menus, returning to the patient selection menu.

Here's the EndEpoch function. It takes the ID of the relevant epoch on the stack, and inserts a duration (oddly enough in milliseconds) in the oLength field.[85]

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (272,
   'COPY->BURY->
   QUERY(SELECT oMade FROM EPOCH WHERE epoch = $[])->FLOAT->
   NOW->FLOAT->SWOP->SUB->
   FLOAT(86400000)->MUL->FLOAT(0.5)->ADD->INTEGER->
   DIGUP->
   DOSQL(UPDATE EPOCH SET oLength=$[] WHERE epoch = $[])',
   'EndEpoch');
```

We find the start timestamp, subtract it from the current time, and multiply by the number of milliseconds in a day. The FLOAT conversions produce Julian days. We round up by adding 0.5 ms.

We continue …

```
INSERT INTO ITEM (iID, iType, iText, iName, iList,
iLines)
          VALUES (701, 2,  'Back',  'bkAl', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (701, 909, 701, 39,
          0.05, 0.900, 0.200, 0.08, 0);
UPDATE ITEM SET iResponse = 'MENU(1)'
  WHERE iID = 701;


INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (731, 2,  'Discharge',  'dsch', '',  1);
```

[85] EndEpoch does not check whether oLength is already populated, not that this should ever be a problem.

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (731, 909, 731, 39,
          0.36, 0.90, 0.28, 0.08, 0, 'red', 'white');
```

Here's the discharge response:

```
UPDATE ITEM SET iResponse =
    '#104->#151->&ProcBetween->
     BOOLEAN->NOT->
     SKIP->=Fail(Please first stop regional!)->
     #389->#391->&ProcBetween->
     BOOLEAN->NOT->
     SKIP->=Fail(Please first stop PCA!)->
     CONFIRM(Alive on discharge?)->
     SKIP->=PatientDied->MENU(DISCHARGE)'
  WHERE iID = 731;
```

Here's *PatientDied*:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (255, 'CONFIRM(Is patient dead?)->
     SKIP->=Fail(Not discharged)->
     $[EpL]->&EndEpoch->
     UNCACHE(EPOCH)->UNCACHE(PROCESS)->
     NOW->X->DOSQL(UPDATE PERSON SET pDied=TIMESTAMP ''$[]'' WHERE person = $[])->
     X->DOSQL(UPDATE BADOBS SET boInactive=1 WHERE Person = $[])->
     X->#0->#9999->&KillManyProcs->MENU(#4)',

  'PatientDied');
```

Note that in the above pDied records the timestamp when we *recorded* the death, which can be a variable period after the actual death. We also invoke EndEpoch to register the time spent processing this 'visit', despite there (usually) being no patient contact!

We terminate all processes associated with the patient, and turn off the active bed status in BADOBS.[86]

We put text Y and N labels at the top of the column of pushbuttons, and then create this column of pushbuttons, with appropriate labels on the left. Items 704, 705 and 706 have been previously defined in the epidural menu!

[86] If we were ever to create processes with codes above 9999, then this code would require modification. Clumsy.

```
-- hypotension:
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (704, 909, 704, 20,
          0.45, 0.20, 0.07, 0.08, 1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (705, 909, 705, 21,
          0.55, 0.20, 0.07, 0.08, 1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (706, 909, 706, 22,
          0.05, 0.20, 0.07, 0.08, 0);
```

For documentation of hypotension, we first need to establish whether there is a 'Blood pressure monitoring process' for this patient (code 1120). If there isn't, once a Y or N button is clicked, we create one. We then determine whether there is an epoch (on this process) which documents either the presence or absence of hypotension.

The *SpawnProblem* routine assumes that $[EpL] exists. It accepts the process type on the stack, and returns the ID of an ISPROBLEM entry, making process, epoch and isproblem entry as required!

We accept the type of the process (eg 1120) on the stack. We MARK the stack below this value, find or create the relevant process, find or create the relevant observation, and then find or create an ISPROBLEM entry. We bury a copy of the problem key, unmark the stack, and then dig up the copy of this key, returning it on the otherwise clean stack.

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (224, 'MARK(#1)->COPY->
    &FindRecentProcess->QOK->SKIP->&NewProc->
    COPY->$[EpL]->
    QMANY(SELECT MAX(EPOCH.epoch) FROM EPOCH WHERE EPOCH.Process = $[]
       AND EPOCH.epoch >= $[])->
    QOK->SKIP->&NewEpoch->
    COPY->
    QUERY(SELECT ISPROBLEM.isproblem FROM ISPROBLEM
          WHERE ISPROBLEM.Epoch = $[])->
    QOK->SKIP->&NewProblem->
    BURY->UNMARK->DIGUP',

  'SpawnProblem');
```

In *SpawnProblem* we check for an EPOCH.epoch greater than *or equal to* EpL because it is also possible to attach an ISPROBLEM entry to EpL and we do

not want to force creation of a new epoch on process 1, or all our references back to EpL will become corrupted!

NewProblem creates a new row in the ISPROBLEM table, given the epoch on the stack. It returns the key of the row on the stack.

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (225, 'BURY->KEY(IsProblem)->COPY->DIGUP->
    DOSQL(INSERT INTO ISPROBLEM(isproblem,Epoch)VALUES($[],$[]))',

  'NewProblem');
```

We also need to be able to fetch the current ISPROBLEM entry:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (226, 'X->&LastEpoch->QOK->SKIP->STOP->
  COPY->$[EpL]->LESS->NOT->SKIP->STOP->
  QUERY(SELECT ISPROBLEM.prIsOrNot FROM ISPROBLEM
          WHERE ISPROBLEM.Epoch = $[])',

  'FetchProblem');
```

FetchProblem uses LESS/NOT rather than GREATER as we entertain the possibility that the problem entry might actually refer to EpL!

SetProblem takes the ISPROBLEM key and a value on the stack (the latter on the stack top) and sets prIsOrNot:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (227, 'SWOP->
    DOSQL(UPDATE ISPROBLEM SET prIsOrNot=$[] WHERE
          isproblem = $[])',

  'SetProblem');

-- sedation
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (710, 4,  'Y',  'sdY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (710, 909, 710, 23,
          0.45, 0.30, 0.07, 0.08, 2);
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (711, 4,  'N',  'sdN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (711, 909, 711, 24,
          0.55, 0.30, 0.07, 0.08, 2);
INSERT INTO ITEM (iID, iType, iText, iName)
```

```
          VALUES (712, 1,  'sedation',  'sdL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (712, 909, 712, 25,
          0.05, 0.30, 0.07, 0.08, 0);

UPDATE ITEM SET iInitial =
       '#1110->&FetchProblem' WHERE iID = 710;
UPDATE ITEM SET iResponse =
       '#1110->&SpawnProblem->#1->&SetProblem'
       WHERE iID = 710;

UPDATE ITEM SET iInitial =
       '#1110->&FetchProblem->QOK->SKIP->#1->BOOLEAN->NOT'
       WHERE iID = 711;
UPDATE ITEM SET iResponse =
       '#1110->&SpawnProblem->#0->&SetProblem'
       WHERE iID = 711;

-- nausea (uses items 707, 708, 709, 716: previously defined)
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (707, 909, 707, 26,
          0.45, 0.40, 0.07, 0.08, 3);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (708, 909, 708, 27,
          0.55, 0.40, 0.07, 0.08, 3);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (709, 909, 709, 28,
          0.05, 0.40, 0.07, 0.08, 0);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (716, 909, 716, 3,
          0.65, 0.40, 0.30, 0.08, 0);

-- pm review
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (713, 3,  '',  'pmr');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (713, 909, 713, 31,
          0.45, 0.10, 0.07, 0.08, 0);
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (715, 1,  'pm review',  'pmL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
```

```
  VALUES (715, 909, 715, 32,
          0.05, 0.10, 0.07, 0.08, 0);

UPDATE ITEM SET iInitial =
       'X->&GetPmFlag->BOOLEAN' WHERE iID = 713;
UPDATE ITEM SET iResponse =
       'SKIP->=EndProcByType(#1100)->&NewProc(#1100)'
       WHERE iID = 713;


-- a few headings, and problems:

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (722, 1,  'ALERTS:',  'alL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (722, 909, 722, 3,
          0.36, 0.01, 0.28, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (723, 1,  'Any substantial problem?',  'qcL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (723, 909, 723, 34,
          0.05, 0.60, 0.50, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (724, 4,  'Y',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (724, 909, 724, 35,
          0.75, 0.60, 0.07, 0.08, 4);
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (725, 4,  'N',  'prN');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (725, 909, 725, 36,
          0.85, 0.60, 0.07, 0.08, 4);

UPDATE ITEM SET iInitial =
       '#1->&FetchProblem' WHERE iID = 724;
UPDATE ITEM SET iResponse =
       '#1->&SpawnProblem->#1->&SetProblem->MENU(COMMENTS)'
       WHERE iID = 724;

UPDATE ITEM SET iInitial =
       '#1->&FetchProblem->QOK->SKIP->#1->BOOLEAN->NOT'
       WHERE iID = 725;
```

```
UPDATE ITEM SET iResponse =
       '#1->&SpawnProblem->#0->&SetProblem'
       WHERE iID = 725;
```

The following item will pull in the most recent comment.

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (259, 909, 159, 3,
          0.05, 0.740, 0.94, 0.08, 1);
```

5.17 Discharge menu (921)

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (921, 20, 'Discharge', 'DISCHARGE');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (1921, 921, 921, 0,
            0.001, 0.001, 0.990, 0.990, 0);

-- the Abort button, used previously:
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1922, 921, 80, 40,
          0.05, 0.900, 0.200, 0.08, 0, 'green', 'white');

-- the 'Confirm Discharge' button:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1923, 2,  'Discharge Now!',  'cbtn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1923, 921, 1923, 41,
          0.50, 0.90, 0.45, 0.08, 0, 'red', 'white');
UPDATE ITEM SET iResponse = '&DoDischarge'
       WHERE iID = 1923;


INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1924, 1,  'REFERRAL TO:',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1924, 921, 1924, 3,
          0.05, 0.200, 0.50, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1925, 1,  'TARPS',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
```

```
  VALUES (1925, 921, 1925, 4,
          0.18, 0.30, 0.50, 0.08, 0);
-- the tick-box:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1935, 3,  '',  'trps');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1935, 921, 1935, 5,
          0.05, 0.30, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '&GetDProc(#2)' WHERE iID = 1935;
UPDATE ITEM SET iResponse = 'SET(tarps)' WHERE iID = 1935;


INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1926, 1,  'Inpatient Palliative Care',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1926, 921, 1926, 6,
          0.18, 0.40, 0.50, 0.08, 0);
-- the tick-box:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1936, 3,  '',  'ippc');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1936, 921, 1936, 7,
          0.05, 0.40, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '&GetDProc(#3)' WHERE iID = 1936;
UPDATE ITEM SET iResponse = 'SET(inpall)' WHERE iID = 1936;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1927, 1,  'Community Palliative Care',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1927, 921, 1927, 8,
          0.18, 0.50, 0.50, 0.08, 0);
-- the tick-box:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1937, 3,  '',  'oppc');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1937, 921, 1937, 9,
          0.05, 0.50, 0.07, 0.08, 0);
UPDATE ITEM SET iInitial = '&GetDProc(#4)' WHERE iID = 1937;
UPDATE ITEM SET iResponse = 'SET(outpall)' WHERE iID = 1937;

-----------------------------------------------------------------------
-- Optional comment + field: item 1805 is also used later!
INSERT INTO ITEM (iID, iType, iText, iName)
```

```
  VALUES (1805, 1,  'Optional comment:',  'mdat');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1938, 921, 1805, 10,
          0.05, 0.60, 0.30, 0.08, 0);

-- the comment field. Re-uses 661. Requires $[comment]
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1939, 921, 661, 11,
          0.05, 0.75, 0.90, 0.08, 0);
```

Here's the initialisation for the Discharge menu (ID 921):

```
UPDATE ITEM SET iInitial =
 'NAME(comment)->
  name(tarps)->name(inpall)->name(outpall)->
  NAME(EpL)->#1->X->&LastEpoch->SET(EpL)->
  #0->set(tarps)->#0->set(inpall)->#0->set(outpall)->
 X->&FetchIdNumber->X->&FetchSurname->Title($[] : $[])'
  WHERE iID = 921;
```

We require EpL as we use this epoch when invoking the EndEpoch routine.

Here's *DoDischarge*. At present it's pretty simple, but we *do* check for various flags associated with referrals, and document these referrals by creating distinct post-discharge processes (process code 5). On readmission, do we check on these? Certainly don't re-create them if they exist on rpt discharge!

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (228, '$[comment]->ISNULL->SKIP->&MakeGeneralComment->
     X->
     DOSQL(UPDATE BADOBS SET boFlag=NULL WHERE Person = $[])->
     $[tarps]->NOT->SKIP->&PostDProc(#2)->
     $[inpall]->NOT->SKIP->&PostDProc(#3)->
     $[outpall]->NOT->SKIP->&PostDProc(#4)->
     $[EpL]->&EndEpoch->
     X->#0->#9999->&KillManyProcs->
     UNCACHE(EPOCH)->UNCACHE(PROCESS)->
     MENU(#5)',
  'DoDischarge');
```

We've increased the number of menus popped to 5 owing to our new format as of 2007-11-14.

First up we check for the presence of a comment and (if present) attach this to the general observation process.

We remember to turn off the active bed status in BADOBS so that we don't 'see' the patient as being admitted on the ward.[87]

We then terminate the current visit's epoch before we end the procedures and turn off CACHING. The MENU(4) command removes all four stacked menus. In a previous incarnation of DoDischarge, we did *not* terminate processes above code 1000, retaining problems such as renal dysfunction as 'active'. There are several problems with such an approach (we aren't monitoring the process; it forces export of the data to the PDA and will eventually clog up the PDA tables) so now we terminate all processes on discharge (up to 9999 anyway!), even the PostDProc!

GetDProc checks whether a given discharge process (associated with a particular virtual person) exists, returning 1 or 0.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (256,
    'X->SWOP->
     QUERY(SELECT process FROM PROCESS WHERE ProcType = 5 AND
      Person = $[] AND rPlanner = $[])->
     QOK->SKIP->RETURN(#0)->DISCARD->
     RETURN(#1)',

   'GetDProc');
```

Finally, here's the 'Post-discharge' process creator, which attaches a particular 'rPlanner' value to a discharge process. The code is clumsily hard-coded above and submitted on the stack:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (257,
    'COPY->X->SWOP->
     QUERY(SELECT process FROM PROCESS WHERE ProcType = 5 AND
      Person = $[] AND rPlanner = $[])->QOK->
      NOT->SKIP->RETURN->
    BURY->KEY(Process)->X->now->now->DIGUP->
    DOSQL(INSERT INTO PROCESS
     (process,Person,rStart,rCreated,rPlanner,ProcType)
     VALUES($[],$[],TIMESTAMP ''$[]'',TIMESTAMP ''$[]'',$[],5))',

   'PostDProc');
```

After checking for the existence of a similar process for this patient, we return if one exists. Otherwise we create the process. We should return nothing but at present this clumsy routine unbalances the stack.

[87] This bed status observation applies solely to our 'pain team' perspective on the patient.

5.18 Help menus

Clicking on the header of a menu is convenient, and we utilise this ability. We permit the attachment of an arbitrary script to any menu thus:

```
UPDATE ITEM SET iResponse = '$[activeW]->ALERT(Wards with flagged patients: $[])'
       WHERE iID = 900;
```

Here's our initialisation script for the same menu, which is now rather complex, to allow us to find out which wards contain cases!

```
update item set iInitial = 'NAME(useW)->NAME(activeW)->#0->
  SET(useW)->MARK(#0)->BURY(#0)->
  QMANY(SELECT DISTINCT WARD.swrdText FROM
  BADOBS,BED,ROOM,WARD WHERE BADOBS.boFlag = 1
  AND BADOBS.boInactive IS NULL AND BADOBS.cold IS NULL
  AND BADOBS.Bed = BED.bed AND BED.Room = ROOM.room
  AND ROOM.Ward = WARD.ward ORDER BY WARD.swrdText)->
  QOK->SKIP->"-"->JOIN(,)->" ,$[],"->SET(activeW)->UNMARK' where iid = 900;
```

...but we might just as easily attach a MENU command within a script, allowing us to enter menus with more details of past observations. In the following pages we illustrate several such menus, for example, to provide details of past pain scores.

5.18.1 Pain help menu

Figure 15: Pain Help menu

```
UPDATE ITEM SET iResponse = 'MENU(PAINHX)'
       WHERE iID = 907;
```

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (950, 20, 'Past Pain Data', 'PAINHX');
-- UPDATE ITEM SET iInitial = '' WHERE iID = 950;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (950, 950, 950, 0,
            0.001, 0.001, 0.990, 0.990, 0);
--- Self-reference.

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1050, 8,  '[No history]',  'PnHx', '',  12);
--- polymenu for pain data

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1054, 1,  '-',  'DtTm', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1053, 1,  '-',  'PnMv', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1052, 1,  '-',  'PnRs', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1051, 1,  '-',  'Cgh', '',  1);

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1054, 1050, 1054, 1, 'Date/time', 0.45);
--- date/time of observation column

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1053, 1050, 1053, 2, 'Mvmt', 0.18);
-- pain on movement

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1052, 1050, 1052, 2, 'Rest', 0.18);
-- rest pain

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1051, 1050, 1051, 2, 'Cough', 0.19);
-- cough

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup, miEnabled)
  VALUES (1050, 950, 1050, 22,
            0.001, 0.01, 0.999, 0.850, 0, 0);
--- above is pain info (1050)

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1060, 2,  'Done',  'done', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
```

```
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1060, 950, 1060, 10,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
UPDATE ITEM SET iResponse = 'MENU(-1)'
  WHERE iID = 1060;
```

We still must initialise the polytable:

```
UPDATE ITEM SET iInitial =
   'X->QMANY(SELECT PAINSCORE.painscore FROM PAINSCORE,EPOCH,PROCESS
     WHERE PAINSCORE.Epoch = EPOCH.epoch AND
     EPOCH.Process = PROCESS.process AND
     PROCESS.Person = $[] ORDER BY PAINSCORE.painscore DESC)'
   WHERE iID = 1050;
```

. . . as well as the individual items:

```
UPDATE ITEM SET iInitial =
   'V->QUERY(SELECT psoCough FROM PAINSCORE WHERE painscore = $[])->
   SKIP->RETURN(-)->"Ok"'
   WHERE iID = 1051;

UPDATE ITEM SET iInitial =
   'V->QUERY(SELECT psoRest FROM PAINSCORE WHERE painscore = $[])'
   WHERE iID = 1052;

UPDATE ITEM SET iInitial =
   'V->QUERY(SELECT psoMovement FROM PAINSCORE WHERE painscore = $[])'
   WHERE iID = 1053;

UPDATE ITEM SET iInitial =
   'V->QUERY(SELECT EPOCH.oMade FROM PAINSCORE,EPOCH WHERE
   PAINSCORE.painscore = $[] AND PAINSCORE.Epoch = EPOCH.epoch)->
   SPLIT( )->BURY->&ShortDate->
   DIGUP->
   SPLIT(:)->DISCARD->
   "$[] $[]:$[]"'
   WHERE iID = 1054;
```

In the above we split the timestamp into date and time (on the intervening space). We then split the time into HH MM SS, discarding the seconds.

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1055, 950, 9926, 99,
          0.05, 0.900, 0.200, 0.08, 0, 'yellow', 'black');
```

5.18.2 PCA help menu

```
UPDATE ITEM SET iResponse = 'MENU(PCAHX)'
       WHERE iID = 914;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (951, 20, 'PCA History', 'PCAHX');
```

Here's the initialisation script for the PCA History menu. Not only do we set up *prQ*, we also determine whether the drug used is fentanyl, and create and set the *fentanyl* variable, so that we can specify micrograms rather than mg.

```
UPDATE ITEM SET iInitial =
  'NAME(prQ)->X->#389->#391->&FindAdministration->SET(prQ)->
   NAME(fentanyl)->#0->SET(fentanyl)->$[prQ]->
    QUERY(SELECT DRUG.dTrade FROM RX,DRUG
      WHERE RX.Process = $[] AND
      RX.Drug = DRUG.drug)->
      IN(Fentanyl)->SET(fentanyl)'
  WHERE iID = 951;
```

The *fentanyl* variable is used by *FromMicrograms*!

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (951, 951, 951, 0,
            0.001, 0.001, 0.990, 0.990, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1070, 8,  '[No PCA history]',  'PnHx', '',  12);
--- polymenu for pain data

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1074, 1,  '-',  'DtTm', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1073, 1,  '-',  'PcaD', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1072, 1,  '-',  'PcaT', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1071, 1,  '-',  'PcTo', '',  1);

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1074, 1070, 1074, 1, 'Date/time', 0.45);
--- date/time of observation column

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1073, 1070, 1073, 2, 'Doses', 0.18);
-- pca doses
```

```
INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1072, 1070, 1072, 2, 'Tries', 0.18);
-- pca tries

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1071, 1070, 1071, 2, 'Tot', 0.19);
-- pca total dose

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup, miEnabled)
  VALUES (1070, 951, 1070, 22,
            0.001, 0.01, 0.999, 0.850, 0, 0);
--- pca info (1070) into menu

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1080, 2,  'Done',  'done', '',  1);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1080, 951, 1080, 10,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
UPDATE ITEM SET iResponse = 'MENU(-1)'
  WHERE iID = 1080;

-- a PCA history button:
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1079, 951, 9926, 99,
          0.05, 0.900, 0.200, 0.08, 0, 'yellow', 'black');
```

We still must initialise the polytable:

```
UPDATE ITEM SET iInitial =
   '$[prQ]->
   QMANY(SELECT epoch FROM EPOCH WHERE Process = $[] ORDER BY epoch)'
   WHERE iID = 1070;
```

. . . as well as the individual items:

```
UPDATE ITEM SET iInitial =
   'V->QUERY(SELECT pcoGood FROM PCA WHERE Epoch = $[])'
   WHERE iID = 1073;

UPDATE ITEM SET iInitial =
   'V->QUERY(SELECT pcoTries FROM PCA WHERE Epoch = $[])'
   WHERE iID = 1072;
```

```
UPDATE ITEM SET iInitial =
   'V->&FindTotal->&FromMicrograms'
   WHERE iID = 1071;

UPDATE ITEM SET iInitial =
   'V->QUERY(SELECT oMade FROM EPOCH WHERE epoch = $[])->
   SPLIT( )->BURY->&ShortDate->
   DIGUP->
   SPLIT(:)->DISCARD->
   "$[] $[]:$[]"'
   WHERE iID = 1074;
```

In the above we split the timestamp into date and time (on the intervening space). We then split the time into HH MM SS, discarding the seconds.

5.18.3 Regional help menu

Note that in the initialisation of this menu we use EpR and not prQ, as EpR is required by the regional query process *FetchEpi*

```
UPDATE ITEM SET iResponse = 'MENU(RGNHX)'
       WHERE iID = 904;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (952, 20, 'Regional Problems (+)', 'RGNHX');
UPDATE ITEM SET iInitial =
  'NAME(EpR)->&FindRegional->SET(EpR)'
  WHERE iID = 952;
-- find regional process (EpR) to which observations will be attached

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (952, 952, 952, 0,
            0.001, 0.001, 0.990, 0.990, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1110, 8,  '[No Regional history]',  'PnHx', '',  12);
--- polymenu for pain data

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1114, 1,  '-',  'DtTm', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1113, 1,  '-',  'RMot', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1112, 1,  '-',  'RPre', '',  1);
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
```

```
          VALUES (1111, 1,  '-',  'RSit', '',  1);
-- motor/pressure areas/site [OK signalled by a -, otherwise if problem "x"]

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1114, 1110, 1114, 1, 'Date/time', 0.45);
--- date/time of observation column

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1113, 1110, 1113, 2, 'Motor', 0.18);
-- Motor

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1112, 1110, 1112, 3, 'Press', 0.18);
-- pca tries

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1111, 1110, 1111, 4, 'Site', 0.18);
-- pca total dose

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup, miEnabled)
  VALUES (1110, 952, 1110, 22,
            0.001, 0.01, 0.999, 0.850, 0, 0);
--- regional info (1110) into menu

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1120, 2,  'Done',  'done', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1120, 952, 1120, 10,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
UPDATE ITEM SET iResponse = 'MENU(-1)'
  WHERE iID = 1120;
```

We still must initialise the polytable:

```
UPDATE ITEM SET iInitial =
   '$[EpR]->
   QMANY(SELECT epoch FROM EPOCH WHERE Process = $[] ORDER BY epoch)'
   WHERE iID = 1110;
```

. . . as well as the individual items:

```
UPDATE ITEM SET iInitial =
   '"Motor"->V->&FetchEpi->COPY->ISNULL->NOT->SKIP->RETURN->SKIP->RETURN(+)->"-"'
   WHERE iID = 1113;
```

A value of ‘1’ returned by FetchEpi signals that there were *no* abnormalities; a value of zero signals a problem. If FetchEpi found null, then NULL is returned.[88]

```
UPDATE ITEM SET iInitial =
   '"Pressure"->V->&FetchEpi->COPY->ISNULL->NOT->SKIP->RETURN->SKIP->RETURN(+)->"-
   WHERE iID = 1112;

UPDATE ITEM SET iInitial =
   '"Site"->V->&FetchEpi->COPY->ISNULL->NOT->SKIP->RETURN->SKIP->RETURN(+)->"-"'
   WHERE iID = 1111;

UPDATE ITEM SET iInitial =
   'V->QUERY(SELECT oMade FROM EPOCH WHERE epoch = $[])->
   SPLIT( )->BURY->&ShortDate->
   DIGUP->
   SPLIT(:)->DISCARD->
   "$[] $[]:$[]"'
   WHERE iID = 1114;
```

Split timestamp into date/time etc.

5.19 Patients without wards

It sometimes happens that a patient is ‘in limbo’, moving between wards; it’s also possible that the person taking down the particulars of a newly referred patient may have not recorded the ward. Finally, it’s common with some computerised referral systems (no names here) to find that the patient’s ward isn’t represented in the ‘referral information’. For all of these reasons, we need to have some sort of ‘staging’ area from which these patients can be moved when their location is identified. We allocate the ‘ward’ with code 1 as this ‘staging area’.

As a solution to the problem of moving these patients to their final destinations, we attach a complete ‘New patient’ menu to a button in the main menu. This menu is fairly simple — it contains a polymorphic menu with the identifiers, and surnames of all such ‘unallocated patients’, with the ability to assign them to a ward using a pop-menu of ward identifiers. Here’s the menu:

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (890, 20, 'New Patients', 'NEWPTS');
```

[88] It’s interesting to see how scripting fails if we say RETURN(.) instead of a simple RETURN in the above (on NULL). This is because the remaining NULL value on the stack interferes with further processing of the row. We need to look at this sort of problem, and protect table drawing against this [FIX ME]! This problem is further justification for creating a REPLACE function or other stack manipulation functions.

```
UPDATE ITEM SET iInitial = 'NAME(id)->NAME(ward)->X->SET(ward)'
   WHERE iID = 890;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (890, 890, 890, 0,
            0.001, 0.001, 0.990, 0.990, 0);
--- Self-reference.

--- Item 401 is a generic menu exit button:
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (9001, 890, 401, 10,
          0.05, 0.91, 0.200, 0.08, 0, 'green', 'white');
--- We also permit an admission button for 'this ward',
--   using our previous 'Admit' button (9927).
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH)
  VALUES (9002, 890, 9927,3, 0.400, 0.91, 0.200, 0.080);

-- We also make use of a former 'more' button (code 9926):
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH)
  VALUES (9003, 890, 9926, 99, 0.75, 0.91, 0.200, 0.080);


--- here's our polymorphic table:
INSERT INTO ITEM (iID, iType, iText, iName, iLines)
  VALUES (2230,8,'[Not found]','nTbl',10);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (2221, 890, 2230,  1, 0.001, 0.001, 0.999, 0.850);

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (2226, 6,  'w',  'Wd', '->&ListWards');
-- this is a poplist

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
                       irName, irFraction, irEnabled)
  VALUES (2226, 2230, 2226, 3, 'Ward',      0.25, 3),
         (2227, 2230, 1227, 2, 'ID',      0.30, 2),
         (2228, 2230, 1228, 1, 'Surname', 0.45, 1);
--- we use items 1227 and 1128 from Menu 918 above.
```

Here's the response to clicking on a Ward (to move a patient to a new ward).

```
UPDATE ITEM SET iResponse  = 'V->&SetNewWard->MENU(#0)'
WHERE iID =2226;
```

In this response, before we invoke *SetNewWard*, we place the ID of the patient on the stack above the ID of the ward. On 2007-11-7 we added in an ALERT to confirm the ward moved to; on 2008-02-25 we made the logical change of confirming the movement before it takes place. We have now moved this confirmation to SetNewWard itself!

Here's the initialisation of the polymorphic table. It's similar to *GetBadobs4Ward* but returns a list of individuals 'allocated' to the staging ward #1.

```
UPDATE ITEM SET iInitial =
 'QMANY(SELECT Person FROM BADOBS WHERE
   cold IS NULL AND
   boInactive IS NULL
   AND Bed <= 10000)'
WHERE iID = 2230;
```

5.20 Reasons for stopping (880)

We really desire to *always* document why PCA or an epidural was stopped. The WHYSTOP and STOPPROC tables are described in *AnalgesiaDBpart1.tex* — here we use them.

The basic scenario is:

1. To stop a regional catheter or PCA, we have a 'Remove PCA' and 'Regional out' buttons, or alternatively we might simply click on the [N] button when the regional/PCA is still recorded in the database as being 'in'.

2. If we click on one of the 'Remove' buttons, we enter a confirmatory menu which allows us to [Abort] or [Confirm] but confirmation requires that the *most important* reason for cessation has been selected from a poplist of reasons. On confirmation, this menu is closed *and* so is the calling menu!

3. If we click on an [N] button, we obtain a similar menu, but there is one difference. This time the calling menu (which is after all the central pain menu) won't be closed. We thus have a separate menu for this option.

Here's the first 'Reason for stopping' menu which closes not only itself but also the calling menu! As X, the transfer variable, we submit the process we are stopping, for example an epidural process.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (880, 20, 'stop', 'WHYSTOP');
UPDATE ITEM SET iInitial = '&SetupStop'
```

```
  WHERE iID = 880;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (880, 880, 880, 0,
            0.001, 0.001, 0.990, 0.990, 0);
--- Self-reference.

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (1800, 6,  '',  'rs',
  '->&GetStopReasons');
UPDATE ITEM SET iResponse = 'SET(rstop)'
    WHERE iID = 1800;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1800, 880, 1800, 3,
          0.05, 0.15, 0.90, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (1801, 1,  'Select reason:',  'mdlbl');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1801, 880, 1801, 4,
          0.05, 0.05, 0.30, 0.08, 0);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1805, 880, 1805, 5,
          0.05, 0.35, 0.30, 0.08, 0);

-- the comment field (item 661 exists already, setting $[comment] )
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1806, 880, 661, 3,
          0.05, 0.45, 0.90, 0.08, 0);

-- date picker (amended 14-11-7):
-- we use item 1105 which requires NAME(rdate):
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1811, 880, 1105, 7,
          0.05, 0.70, 0.35, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (1812, 1,  'Date stopped: (default TODAY)',  'ds');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
```

```
          miX, miY, miW, miH, miGroup)
  VALUES (1812, 880, 1812, 8,
          0.05, 0.60, 0.50, 0.08, 0);


-- abort button:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1807, 2,  'Abort',  'abrt', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1807, 880, 1807, 10,
          0.05, 0.900, 0.200, 0.08, 0, 'green', 'white');
UPDATE ITEM SET iResponse = 'MENU(#1)'
  WHERE iID = 1807;

-- terminate process:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1810, 2,  'STOP',  'ok', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1810, 880, 1810, 11,
          0.750, 0.900, 0.200, 0.08, 0, 'red', 'white');
```

The response to clicking on 'STOP' is now fairly complex. We first record the reasons for stopping, and we then check to see whether 24 hours have elapsed since stopping an epidural process (this is because it is possible that the date recorded for stopping an epidural is not the current date). If 24 hours *have elapsed*, we skip the MENU(#2) command which otherwise discards this and the preceding (regional or pca) menu. Once we've skipped, we pop *three* menus, the current, preceding and main pain menu, and enter the RGNCHECK menu. Ultimately, leaving the RGNCHECK menu will take us to the main menu once more. Note that X is the process being stopped, hence the modification to *Is24*.

```
UPDATE ITEM SET iResponse ='&RecordStopReasons->X->
   QUERY(SELECT Person FROM PROCESS WHERE process = $[])->
   COPY->SetX->
   &Is24->BOOLEAN->SKIP->MENU(#2)->
   MARK(#0)->POPMENU(#2)->POPMENU(#1)->POPMENU(#0)->UNMARK->MENU(RGNCHECK)'
  WHERE iID = 1810;
```

The above if finicky — we have to SetX *in case* we enter the RGNCHECK menu! (The alternative would be to set one of the X values obtained from POPMENU).

We initialise the menu with the routine *SetupStop*:

```
INSERT INTO FUN (fKey, fBody, fName)
```

```
    VALUES (278,
   'NAME(rdate)->NOW->SPLIT( )->DISCARD->SET(rdate)->
    NAME(patient)->NAME(proctype)->
   X->QUERY(SELECT Person,ProcType FROM PROCESS WHERE process = $[])->
   SET(proctype)->SET(patient)->
   NAME(comment)->NULL->SET(comment)->
   NAME(rstop)->NULL->SET(rstop)->
   NAME(EpL)->#1->$[patient]->&LastEpoch->SET(EpL)->
   "Other Rx"->
    $[proctype]->GREATER(#100)->$[proctype]->LESS(#151)->AND->NOT->SKIP->REPLACE(R
    $[proctype]->SAME(#390)->NOT->SKIP->REPLACE(PCA)->
    $[proctype]->SAME(#50)->NOT->SKIP->REPLACE(all orals)->
   "STOP $[]?"->TITLE',
 'SetupStop');
```

In the above we set up the comment, rdate, rstop and proctype variables. We also find the most recent epoch for the general observation process on this person, and move the value to the local variable 'EpL'. This variable is used by

Here's the simple *GetStopReasons* which merely retrieves an appropriate list of pairs — each pair consisting of a stop code and the associated text reason.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (276,
    'QMANY(SELECT whystop,wText FROM WHYSTOP)',
 'GetStopReasons');
```

RecordStopReasons is more tricky. We associate the major reason for stopping with the process in X, but also terminate this process, and any similar processes. The latter is a little taxing, as different processes will be associated depending on whether we're stopping a regional infusion, PCA, or other process.

We rely on values in the local variables rstop and proctype, which must be set up prior to invocation, as must the local variable 'comment'!

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (277,
    '$[rstop]->ISNULL->NOT->SKIP->=Fail(Please choose a reason!)->
    KEY(StopProc)->X->$[rstop]->
    DOSQL(INSERT INTO STOPPROC(stopproc,Process,Whystop)
      VALUES($[],$[],$[]))->
    $[proctype]->GREATER(#100)->$[proctype]->LESS(#151)->AND->NOT->SKIP->&JustKill
    $[proctype]->GREATER(#259)->$[proctype]->LESS(#300)->AND->NOT->SKIP->&JustKill
    $[proctype]->SAME(#390)->NOT->SKIP->&JustKillPca->
    $[proctype]->SAME(#50)->NOT->SKIP->&JustKillOrals->
    $[comment]->ISNULL->NOT->SKIP->RETURN->
    KEY(Comment)->X->&ForceEpoch->$[comment]->&FixSQL->
    DOSQL(INSERT INTO COMMENT(comment,Epoch,cText)
      VALUES($[],$[],''$[]''))',
 'RecordStopReasons');
```

ForceEpoch will create a brand new epoch if none exists for the given process. This is distinctly unusual, but can occur if e.g. a new oral drug is added, and then all drugs are cancelled! It would be more honest in the above to attach a comment to the general (type 1) process than to a particular oral or other drug, when we terminate all.

JustKillRegional allows us to *turn off and remove* the regional infusion. It requires **rdate**.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (178,
    '$[patient]->#98->#151->&KillDated->
    $[patient]->#199->#251->&KillDated->
    $[patient]->#299->#351->&KillDated->
    &SetNonevent(#1)->
    ALERT(Regional terminated)->
    $[proctype]->SAME(#110)->SKIP->RETURN->
    $[patient]->#99->$[rdate]->"$[] 00:00:00"->&DatedProc',
 'JustKillRegional');
```

On 22/1/2008 we fixed *JustKillRegional* so that the date used in establishing the type 99 reminder process is the date of cessation of the regional, and *not* the current date.

The three invocations of *KillManyProcs* are needed to remove all instances of regional catheters and regional infusions (including infusions with PCA, codes 300–350)! We also set up a type 99 process (using *DatedProc*, which ensures that later on (the next day?) we perform a check on the results of the regional.

We amended this routine on 2007-11-07 so that the type 99 check doesn't pop up for anything other than an epidural.[89] This requires use of the $[proctype] variable by *JustKillRegional*.

JustKillPca is analogous to *JustKillRegional*. It requires rdate, the date of stopping (but no time: at present we won't be that precise)!

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (217,
    '$[patient]->#389->#391->&KillDated->
    &SetNonevent(#2)->
    ALERT(PCA terminated)',
 'JustKillPca');
```

JustKillOrals is analogous to *JustKillPca*. On confirmation *all* current oral processes are terminated!

[89] It seems unnecessary and caused a fuss with e.g. sciatic catheters.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (219,
    '$[patient]->#49->#51->&KillManyProcs->
    &SetNonevent(#3)->
    ALERT(All orals stopped!)',
 'JustKillOrals');
```

Note that at present this routine does not use **rdate**.

Here's *JustKillOther*, similar to *JustKillOrals*. It can be invoked by clicking on the relevant [N] button. It too does not require **rdate** at present!

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (230,
    '$[patient]->#259->#300->&KillManyProcs->
    &SetNonevent(#4)->
    ALERT(Modalities stopped!)',
 'JustKillOther');
```

Here's a related routine, *ConsiderStopping*. It takes a range of process codes, and if a process is identified in this range, enters the WHYSTOP menu.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (279,
    '&ProcBetween->
     COPY->BOOLEAN->SKIP->RETURN->
     SETX->MENU(WHYSTOP)',
 'ConsiderStopping');
```

See how we pass the *process* to the WHYSTOP menu as the transfer variable X!

5.20.1 The second 'stop' menu

Initially we created a second menu, identical to the above, but accessible by a click on the [N] button of an existing item. We have replaced this with the following hacks, intended to compensate for our as yet unresolved problem with menu invocation from the [N] pushbutton.

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (280,
    'BURY->&ProcBetween->DIGUP->SWOP->
     BOOLEAN->NOT->SKIP->=HackB->
     &SetNonevent',
 'HackA');
```

```
INSERT INTO FUN (fKey, fBody, fName)
    VALUES (281,
    'COPY->SAME(#1)->NOT->SKIP->REPLACE(Regional)->
     COPY->SAME(#2)->NOT->SKIP->REPLACE(PCA)->
     COPY->SAME(#3)->NOT->SKIP->REPLACE(Oral Rx)->
     COPY->SAME(#4)->NOT->SKIP->REPLACE(Miscellaneous Rx)->
     ALERT($[] exists! You might wish to view this. Click on [Y], not [N]!)',
 'HackB');
```

5.21 Logging in (899)

Figure 16: Logging in

We need to identify the current user. Although this simple screen will actually be the first one, in our documentation we've left it until last!

At present we do NOT require input of a password or other identification, simply asking "Are you such-and-such".

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (899, 20, 'Honest log-in', 'MAIN');
UPDATE ITEM SET iInitial = 'NAME(me)->NULL->SET(me)'
   WHERE iID = 899;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (899, 899, 899, 0,
            0.001, 0.001, 0.990, 0.990, 0);
--- Self-reference.

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (897, 1,  'Your name?',  '');
```

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (897, 899, 897, 3,
          0.03, 0.02, 0.25, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (896, 6,  '',  'User', '->&ListUsers');
UPDATE ITEM SET iInitial = '?' WHERE iID = 896;
UPDATE ITEM SET iResponse ='SET(me)' WHERE iID = 896;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (896, 899, 896, 3,
          0.03, 0.12, 0.85, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (898, 2,  'Enter',  'ntr', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (898, 899, 898, 10,
          0.400, 0.600, 0.200, 0.08, 0, 'green', 'white');

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (894, 1,  '',  '');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (894, 899, 894, 3,
          0.03, 0.90, 0.55, 0.08, 0);

UPDATE ITEM SET iInitial = 'NOW->"Date and time: $[]"' WHERE iID = 894;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (893, 2,  'Bad date/time (fix)',  '');

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (893, 899, 893, 10,
          0.10, 0.800, 0.80, 0.08, 0, 'red', 'white');

UPDATE ITEM SET iResponse = 'MENU(FIXTIME)' WHERE iID = 893;

-- and an 'about' button:

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (895, 2,     '?',  '');
```

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (895, 899, 895, 3,
          0.90, 0.01, 0.05, 0.08, 0);

UPDATE ITEM SET iResponse = 'ALERT(PainForm v 0.95. Copyright (C) J van Schalkwyk
```

It's a good idea to display the current date and time, and to insert an 'Abort' button for the user who discovers that the date/time is incorrect. Perhaps have instructions about how to alter this on the PDA!

Here's the crucial response to pressing 'Enter':

```
UPDATE ITEM SET iResponse =
  '$[me]->ISNULL->NOT->SKIP->=Fail(Please say who you are!)->
  $[me]->Confirm(Is your ID No. $[]?)->SKIP->STOP->
  $[me]->INTEGER->SETME->
  &FlagRecent->
  MENU(MAIN2)'
  WHERE iID = 898;
```

Now for the function which lists all potential users:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (245,
        '#0->
        QMANY(SELECT PERSON.person,PERSDATA.pdoForename,PERSDATA.pdoSurname
        FROM PERSDATA,EPOCH,PROCESS,PERSON WHERE
        PERSDATA.Epoch = EPOCH.epoch AND
        EPOCH.Process = PROCESS.process AND
        PROCESS.Person = PERSON.person AND
        PROCESS.rEnd IS NULL AND
        PERSON.pStatus > 1)->
        #0->BURY->
        REPEAT(&ProcessUserNames)->DISCARD->
        REPEAT(&Retrieve)',

        'ListUsers');
```

The above is mildly in error as it assumes that the PERSDATA table entry is unique for each person (We might make this mandatory for the PDA, but on the desktop it is possible that e.g. the surname might have changed). We also don't use the denormalised pdoPerson field, which would simplify things (but introduce hidden users with an rEnd which is not null). We not only ignore patients (pStatus = 1) but also ignore 'virtual' people (used to signal e.g. referral to another discipline on discharge) who have a pStatus of zero.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (246,
        'COPY->SAME(#0)->NOT->SKIP->STOP->
        "$[] $[]"->BURY->BURY',

        'ProcessUserNames');
```

In the above, we have to generate *pairs* of data: one component of the pair is a text string containing forename and surname, the second is the unique ID of that person. The surname and forename are on the top of the stack so we compound and bury these before burying the ID.

5.22 An Introductory Screen (970)

On rolling out the database, some users found that the PDA screens above are a bit inscrutable, and wanted a better overview of what's going on. This seems reasonable, and the INTRO menu is designed to address this need. This menu is read-only, apart from the facility for moving the patient to another ward. It has the following contents:

1. The most recent operation date (in short form e.g. 22/7), and the operation description (but no type);
2. Entries for EPIdural and PCA. If these are active, we say e.g. D3; if historical, we give a short date range e.g. 22/7–25/7.
3. Tick boxes to indicate whether a problem has been flagged for the patient in the last twenty four hours, and whether they are on ketamine;
4. Four lines for the four most recent comments
5. A pop-menu to allow changing of the ward
6. Buttons to go Back or 'Add Data'.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (970, 20, 'Summary', 'INTRO');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (970, 970, 970, 0,
            0.001, 0.001, 0.990, 0.990, 0);
```

We need to intialise this menu, which always accepts the patient ID as 'X', the transfer variable.

```
UPDATE ITEM SET iInitial =
 'X->&FetchIdNumber->X->&FetchSurname->TITLE($[] : $[])->
  NAME(ward)->
  NAME(id)->X->SET(id)'
       WHERE iID = 970;
```

The above caching was formerly started within *EnterDetailMenu*. The [id] variable is a hack, to satisfy ReadmitPatient. [FIX ME]. We now start caching sooner, so our back button must UNCACHE.

```
-- a Back button
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1970, 2,  'Back',  'cbtn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1970, 970, 1970, 39,
          0.05, 0.910, 0.200, 0.08, 0, 'green', 'white');
UPDATE ITEM SET iResponse = 'UNCACHE(EPOCH)->UNCACHE(PROCESS)->MENU(1)'
       WHERE iID = 1970;

-- the 'Add data' button:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1971, 2,  'Add data..',  'cbtn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1971, 970, 1971, 41,
          0.50, 0.91, 0.45, 0.08, 0, 'red', 'white');
UPDATE ITEM SET iResponse = '&EnterDetailMenu'
       WHERE iID = 1971;

-- Operation (Op:)
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1972, 1,  'No operation documented',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1972, 970, 1972, 1,
          0.01, 0.20, 0.98, 0.08, 1);
-- group of 1 means 'clickable'
```

If an operation exists, we replace 'No operation' as follows.

```
UPDATE ITEM SET iInitial =
  '&LastOp->QOK->SKIP->RETURN->
   "$[]: $[]"' WHERE iID = 1972;
```

Here's the routine to determine the most recent operation date and name. If the SQL fails we return nothing. Callers depend on QOK.

```
 INSERT INTO FUN (fKey, fBody, fName)
   VALUES (290,
    'X->QMANY(SELECT MAX(process) FROM PROCESS
  WHERE PROCESS.Person = $[] AND
  PROCESS.ProcType = 500)->QOK->SKIP->RETURN->
  COPY->
  QUERY(SELECT COMMENT.cText FROM
  COMMENT,EPOCH WHERE
  COMMENT.Epoch = EPOCH.epoch AND
  EPOCH.Process = $[])->QOK->SKIP->"unspecified procedure"->BURY->
  QUERY(SELECT rStart FROM PROCESS WHERE process = $[])->
  SPLIT( )->DISCARD->&ShortDate->
  DIGUP',

  'LastOp');

UPDATE ITEM SET iResponse =
  '&LastOp->QOK->SKIP->RETURN->
   SWOP->DISCARD->Alert'
 WHERE iID =1972;
```

We will have a 'regional' line which specifies the type of regional (if used), and either the *day* e.g. D3, or, if already removed, the range of short dates during which the most recent regional was in.

```
-- Rgnl: (? e.g. 'Epi' ...)

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1973, 1,  'No regional',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1973, 970, 1973, 2,
          0.01, 0.30, 0.90, 0.08, 0);

UPDATE ITEM SET iInitial =
  'X->QMANY(SELECT MAX(process) FROM PROCESS
  WHERE PROCESS.Person = $[] AND
  PROCESS.ProcType > 101 AND
  PROCESS.ProcType < 159)->QOK->SKIP->RETURN->
  QUERY(SELECT PROCTYPE.rptNature,
  PROCESS.rStart,PROCESS.rEnd
  FROM PROCESS,PROCTYPE WHERE
  PROCESS.process = $[] AND
  PROCESS.ProcType = PROCTYPE.proctype)->
  &TinyStamp->COPY->ISNULL->NOT->SKIP->REPLACE(now)->BURY->
  &TinyStamp->
  DIGUP->"$[]: $[]--$[]"' WHERE iID = 1973;
```

In the above we find (the last) putative regional and if it exists, get start and possibly end times. We convert the start time to a short date, and likewise for the end time if it exists.

```
-- PCA:

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1976, 1,  'No PCA',  'pca');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1976, 970, 1976, 2,
          0.01, 0.40, 0.10, 0.08, 0);
```

The following initialisation of the PCA might share a common routine with the regional one:

```
UPDATE ITEM SET iInitial =
  'X->QMANY(SELECT MAX(process) FROM PROCESS
  WHERE PROCESS.Person = $[] AND
  PROCESS.ProcType = 390)->QOK->SKIP->RETURN->
  QUERY(SELECT PROCESS.rStart,PROCESS.rEnd
  FROM PROCESS WHERE PROCESS.process = $[])->
  &TinyStamp->COPY->ISNULL->NOT->SKIP->REPLACE(now)->BURY->
  &TinyStamp->
  DIGUP->"PCA: $[]--$[]"' WHERE iID = 1976;

-- Ketamine tickbox (ever)

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1974, 1,  'Ketamine',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1974, 970, 1974, 3,
          0.01, 0.50, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1984, 3,  '',  'ket');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miEnabled)
  VALUES (1984, 970, 1984, 4,
          0.28, 0.50, 0.07, 0.08, 0, 0);

 UPDATE ITEM SET iInitial =
  'X->QUERY(SELECT PROCESS.process FROM RX,PROCESS WHERE
    RX.Drug = 5001 AND
    RX.Process = PROCESS.process
    AND PROCESS.Person = $[])->QOK->SKIP->RETURN(#0)->
    BOOLEAN'
 WHERE iID = 1984;
```

– this might be a little slow ?

```
-- Problem(s) tickbox (past 48 hr?)

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1975, 1,  'Problems',  'mp');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1975, 970, 1975, 5,
          0.50, 0.50, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1985, 3,  '',  'prb');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miEnabled)
  VALUES (1985, 970, 1985, 6,
          0.75, 0.50, 0.07, 0.08, 0, 0);

UPDATE ITEM SET iInitial =
  '#1->X->&LastEpoch->
  QUERY(SELECT ISPROBLEM.prIsOrNot FROM ISPROBLEM
   WHERE ISPROBLEM.Epoch = $[])' WHERE iID = 1985;
```

We've also moved various 'FYI' fields from the previous 'Alert screen' to here:

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (161, 1,  'Wt(kg)',  'wtl');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (161, 970, 161, 18,
          0.28, 0.01, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (162, 1,  '',  'wt');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (162, 970, 162, 19,
          0.48, 0.01, 0.20, 0.08, 0);
UPDATE ITEM SET iInitial = '&FetchWeight' WHERE iID = 162;


INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (163, 1,  'Age(yr)',  'ay');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (163, 970, 163, 3,
          0.63, 0.01, 0.20, 0.08, 0);
```

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (164, 1,  '',  'yr');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (164, 970, 164, 3,
          0.87, 0.01, 0.20, 0.08, 0);
UPDATE ITEM SET iInitial = '&FetchAge' WHERE iID = 164;

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (134, 1,  'ASA:',  'Asa');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (35, 970, 134, 3,
          0.03, 0.010, 0.15, 0.08, 0);
UPDATE ITEM SET iInitial = '&FetchASA->"ASA $[]"' WHERE iID = 134;
-- asa rating

INSERT INTO ITEM (iID, iType, iText, iName, iLines)
          VALUES (131, 1,  'Given:',  'SurL', 1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (36, 970, 131, 2,
          0.35, 0.111, 0.75, 0.08, 0);
-- forename

UPDATE ITEM SET iInitial =
  'X->&FetchForename->"Given: $[]"' WHERE iID = 131;

INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (138, 6,  '',  'Wd', '->&ListWards');
UPDATE ITEM SET iInitial = '&GetMyWard' WHERE iID = 138;
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (38, 970, 138, 3,
          0.03, 0.111, 0.29, 0.08, 0);
```

Here's the response to changing the ward:

```
UPDATE ITEM SET iResponse = 'X->&SetNewWard' WHERE iID = 138;
```

The above confirmation might profitably be moved to within SetNewWard!
The following displays the 3 most recent comments:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1977, 8,  '[No comments]',  'C', '',  4);
-- 3 lines + header!
```

```
INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1777, 1977, 644, 1, 'Date', 0.25);
--- date of epoch column (we discard time component of timestamp)

INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder, irName, irFraction)
    VALUES (1778, 1977, 645, 2, 'Comment', 0.75);

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup, miEnabled)
  VALUES (1978, 970, 1977, 22,
            0.001, 0.57, 0.999, 0.30, 1, 0);
-- group of 1 forces all comments to be clickable!!
```

Here's the initialisation function:

```
UPDATE ITEM SET iInitial =
       'X->QMANY(SELECT COMMENT.comment FROM COMMENT,EPOCH,PROCESS
          WHERE COMMENT.Epoch = EPOCH.epoch AND EPOCH.Process = PROCESS.process
          AND PROCESS.ProcType <> 500
          AND PROCESS.Person = $[] ORDER BY COMMENT.comment DESC)'
       WHERE iID = 1977;
```

We exclude comments on surgery (process type 500)!

We also have an added [All comments] button:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1158, 2,  'All comments',  'c', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (1979, 970, 1158, 23,
          0.6, 0.59, 0.38, 0.07, 0);

-- Ward alteration [? also bed?]

-- that's it!
```

5.23 Error documentation: 980

There are potential medicolegal implications to documentation of errors as part of an activity which is not necessarily legally protected, but these are outweighed (in my opinion anyway) by the advantages which might potentially accrue to patients if errors are clearly recorded and ultimately acted upon to change the system in which they occurred.

We provide the opportunity for clinicians to record errors simply. First consult the document *AnalgesiaDBpart1.tex* to view the structure of the PAINERROR menu, and then return here.

5.23.1 The PROCERROR menu: 960

Our initial PROCERROR menu will simply list processes for a particular patient. Initially we will limit these to processes with a ProcType under 999, encompassing all processes apart from the high processes associated with systemic problems etc.[90]

We create a menu, and within it have a table of short dates, process names, and associated drugs (if present). We order by date (DESC) and perhaps also by process id.

```
-- the menu itself:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (960, 20, 'Select error domain', 'PROCERROR');
UPDATE ITEM SET iInitial = ''
  WHERE iID = 960;
-- in the above should get&show NHI
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (960, 960, 960, 0,
            0.001, 0.001, 0.990, 0.990, 0);
```

Here simply have big polymorphic table; Back/Abort and More buttons.

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (962, 1,     'Date',    'dt' ),
                 (963, 2,     'What',    'wh' ),
                 (964, 1,     'Drug',    'dr' );

-- create polymorphic table
INSERT INTO ITEM (iID, iType, iText, iName, iLines)
  VALUES (965,8,'[No process!]','prTbl',8);

-- insert table into menu
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH)
  VALUES (965 ,960, 965, 0, 0.001, 0.09, 0.99, 0.80);

-- make table columns
INSERT INTO ICOLTABLE (irKey, irTBL, irItem, irOrder,
                       irName, irFraction, irEnabled)
  VALUES (962, 965, 962, 1, 'Date',      0.15, 0),
         (963, 965, 963, 2, '  What',      0.45, 1),
         (964, 965, 964, 3, 'Drug',      0.40, 0),
```

[90] We might consider adding *all* processes.

```
-- [back] button: uses old button #201
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (966, 960, 201, 7,
          0.01, 0.91, 0.16, 0.08, 0, 'red', 'white');

-- 'more' button: (use old #9926)
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (967, 960, 9926, 99,
          0.75, 0.91, 0.200, 0.08, 0, 'yellow', 'black');
```

We initialise the table:

```
UPDATE ITEM SET iInitial =
  'X->QMANY(SELECT process FROM PROCESS WHERE Person = $[] AND ProcType < 1000)'
  WHERE iID = 965;
```

Get the process type description:

```
UPDATE ITEM SET iInitial = 'V->QUERY(SELECT PROCTYPE.rptNature FROM PROCTYPE,PROCE
  WHERE PROCTYPE.proctype = PROCESS.Proctype AND
        PROCESS.process = $[])' WHERE iID = 963;
```

Create a response to clicking on a given process:

```
UPDATE ITEM SET iResponse = 'V->SETX->MENU(ERR)' WHERE iID = 963;
```

Get the process date:

```
UPDATE ITEM SET iInitial = 'V->QUERY(SELECT rStart FROM PROCESS
  WHERE process = $[])->SPLIT( )->DISCARD->&TinyStamp' WHERE iID = 962;
```

Get (any) associated drug:

```
UPDATE ITEM SET iInitial = 'V->QUERY(SELECT DRUG.dTrade FROM DRUG,RX
  WHERE DRUG.drug = RX.Drug AND RX.Process = $[])->QOK->SKIP->"-"->RETURN'
WHERE iID = 964;
```

A click on the button describing the process invokes the ERR menu with that process as X.

5.23.2 The ERR menu

We will submit as X, the transfer variable, the ID (key) of a particular PROCESS.

```
-- the menu itself:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (980, 20, 'Record error', 'ERR');
```

Let's set up some local variables, and find the most recent epoch for this process now! We assume X is the process.

```
UPDATE ITEM SET iInitial = 'NAME(err)->NAME(date)->NAME(note)->
  NAME(EpL)->NAME(procstart)->NAME(procend)->NAME(isps)->NAME(ispe)->
  #0->SET(isps)->#0->SET(ispe)->
  X->QMANY(SELECT MAX(epoch) FROM EPOCH
     WHERE Process = $[])->QOK->SKIP->#0->SET(EpL)->
  $[EpL]->NOT->SKIP->RETURN->
  X->&NewEpoch->SET(EpL)'
  WHERE iID = 980;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (980, 980, 980, 0,
            0.001, 0.001, 0.990, 0.990, 0);

-- text label:
INSERT INTO ITEM (iID, iType, iText, iName)
         VALUES (780, 1,  'Type',  '');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (780, 980, 780, 1,
          0.03, 0.05, 0.20, 0.08, 0);

-- list of errors:
INSERT INTO ITEM (iID, iType, iText, iName, iList)
  VALUES (781, 6,  '',  'et', '->&ListErrs');
UPDATE ITEM SET iInitial = '?' WHERE iID = 781;
UPDATE ITEM SET iResponse ='SET(err)' WHERE iID = 781;
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (781, 980, 781, 2,
          0.28, 0.05, 0.65, 0.08, 0);

-- date+its label: 782/783 (order 3/4)
INSERT INTO ITEM (iID, iType, iText, iName)
         VALUES (782, 1,  'Date of error',  '');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
```

```
  VALUES (782, 980, 782, 3,
          0.03, 0.25, 0.30, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (783, 12,  '',  'd');
-- type is 12 for date picker!
UPDATE ITEM SET iInitial =
  'NOW->SPLIT( )->DISCARD->COPY->SET(date)' WHERE iID = 783;
UPDATE ITEM SET iResponse = 'SET(date)' WHERE iID = 783;
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (783, 980, 783, 4,
          0.40, 0.25, 0.40, 0.08, 0);

-- mandatory comment+label: 784/785 (order 5/6)
INSERT INTO ITEM (iID, iType, iText, iName)
         VALUES (784, 1,  'Comment:',  '');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (784, 980, 784, 5,
          0.03, 0.34, 0.20, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (785, 10,  '',  'ctxt', '',  1);
UPDATE ITEM SET iResponse = 'set(note)' WHERE iID = 785;

INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (785, 980, 785, 5,
          0.03, 0.42, 0.94, 0.08, 0);

-- [abort] button: 786 (order 7) uses old button #80
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (786, 980, 80, 7,
          0.05, 0.90, 0.200, 0.08, 0, 'red', 'white');

-- [Report error] button (order 8)
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (787, 2,  'Report error',  'ntr', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (787, 980, 787, 10,
          0.50, 0.90, 0.400, 0.08, 0, 'green', 'white');
UPDATE ITEM SET iResponse = '&ReportError' WHERE iID = 787;
```

ReportError is a tad clumsy, as it assumes the existence of the variables err, date and note, all of which must be non-null. (Date will by default be today). If

valid, we insert a record of the error, including the current EPOCH, which must be present in $[EpL].

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (292,
    '$[err]->ISNULL->NOT->SKIP->=Fail(Select an error)->
     $[note]->ISNULL->NOT->SKIP->=Fail(Please insert mandatory comment!)->
     $[note]->$[date]->
     CONFIRM(Document error ''$[]'' on date $[]?)->SKIP->RETURN->
     KEY(Painerror)->$[err]->$[date]->$[note]->&FixSQL->$[EpL]->
     DOSQL(INSERT INTO PAINERROR(painerror,Errtype,ErrStamp,peText,Epoch)
       VALUES($[],$[],TIMESTAMP ''$[] 00:00:00'',''$[]'',$[]))->
     ALERT(Error recorded!)->MENU(1)',
        'ReportError');
```

Here's the *ListErrs* routine that lists all potential error types. As usual for pop-lists, we retrieve the code *and* the text.

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (293,
   'QMANY(SELECT errtype,etText FROM ERRTYPE WHERE cold IS NULL)',
        'ListErrs');
```

It's attractive here to permit actual alterations to process start and end dates. We will limit such alterations to processes with a ProcType between 50 and 500 (inclusive), and must ensure that the rEnd is not null. In addition, we will only permit alteration of start dates if we are on the PDA with a temporary process key value (over 8999999999).

```
INSERT INTO ITEM (iID, iType, iText, iName)
  VALUES (793, 12,  '',  'd');
-- type is 12 for date picker!
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (793, 980, 793, 4,
          0.40, 0.60, 0.40, 0.08, 0);

-- start date label:
INSERT INTO ITEM (iID, iType, iText, iName)
         VALUES (794, 1,  'Process START',  '');
UPDATE ITEM SET iInitial = '$[isps]->SKIP->STOP' WHERE iID = 794;
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (794, 980, 794, 3,
          0.03, 0.60, 0.30, 0.08, 0);
```

The intialisation is rather intricate, as we only accept alterations to the *start* if the process is local to the PDA:

```
UPDATE ITEM SET iInitial =
  'X->GREATER(#899999999)->SKIP->STOP->
  X->QUERY(SELECT rStart FROM PROCESS WHERE process = $[] AND
    ProcType > 49 AND ProcType < 501)->
  QOK->SKIP->STOP->SPLIT( )->DISCARD->COPY->SET(procstart)->#1->SET(isps)'
WHERE iID = 793;
```

5.23.3 Error buttons

We still need to insert error buttons in the relevant menus. Let's try the PCA menu first:

```
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (788, 2,  'Oops',  'ntr', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (788, 914, 788, 10,
          0.50, 0.90, 0.18, 0.08, 0, 'red', 'white');
UPDATE ITEM SET iResponse = 'MENU(PROCERROR)'
  WHERE iID = 788;
```

We can put the identical button in five other menus:

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (789, 904, 788, 10,
          0.50, 0.90, 0.18, 0.08, 0, 'red', 'white');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (790, 915, 788, 10,
          0.50, 0.90, 0.18, 0.08, 0, 'red', 'white');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (791, 2915, 788, 10,
          0.50, 0.90, 0.18, 0.08, 0, 'red', 'white');
-- "ALERTS:" menu (top right part)
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (792, 909, 788, 10,
          0.80, 0.02, 0.18, 0.08, 0, 'red', 'white');
-- and into the entry menu too:
  INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (152, 903, 788, 50,
          0.40, 0.91, 0.18, 0.08, 0, 'red', 'white');
```

Finally, here's the routine to *ensure* we submit a valid epoch to the ERR menu. We assume the existence of EpQ and prQ, and a valid number in prQ.

[NO. DELETE ME]

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (294,
   '$[EpQ]->SAME(#0)->SKIP->RETURN->$[prQ]->&NewEpoch->SET(EpQ)',
        'EnsureEpoch');
```

5.24 Frills

5.24.1 Alter the date (991)

Here we create a menu which permits the user to change the date and time without leaving PainForm.

```
-- the menu itself:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (991, 20, 'Fix Time/Date', 'FIXTIME');
UPDATE ITEM SET iInitial = 'NAME(h1)->NAME(h2)->NAME(m1)->NAME(m2)->NAME(date)'
   WHERE iID = 991;
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
            miX, miY, miW, miH, miGroup)
    VALUES (991, 991, 991, 0,
            0.001, 0.001, 0.990, 0.990, 0);
```

Next, Change and Abort buttons:

```
-- 80 is previously defined 'Abort' button
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1991, 991, 80, 1,
          0.05, 0.90, 0.25, 0.08, 0, 'green', 'white');

INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (1992, 2,  'Set Time',  'a', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (1992, 991, 1992, 2,
          0.75, 0.90, 0.25, 0.08, 0, 'red', 'white');

UPDATE ITEM SET iResponse =
  '$[date]->$[h1]->$[h2]->$[m1]->$[m2]->
  "$[] $[]$[]:$[]$[]:00"->COPY->TIMESTAMP->
  SETTIME->SKIP->=Fail(Bad date/time: $[]!)->MENU(#1)' WHERE iID = 1992;
```

Finally, the date, hour and minute digit buttons:

```
-- 'Date' label:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1994, 1,   'Date','d' );
```

```
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1994, 991, 1994, 3,
          0.03, 0.25, 0.40, 0.08, 0);

-- date box (itype 12 = date picker)
INSERT INTO ITEM (iID, iType, iText, iName, iLines)
          VALUES (1993, 12,  '',  'db', 1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1993, 991, 1993, 4,
          0.35, 0.25, 0.30, 0.08, 0);
UPDATE ITEM SET iInitial = 'NOW->SPLIT( )->DISCARD->COPY->SET(date)' WHERE iID = 1
UPDATE ITEM SET iResponse = 'SET(date)' WHERE iID = 1993;

-- 'time' label
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1995, 1,   'Time', 't' );
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1995, 991, 1995, 5,
          0.03, 0.50, 0.30, 0.08, 0);

-- individual poplists H H : M M

INSERT INTO ITEM (iID, iType, iText, iList)
          VALUES (1996, 6,     '?', '0|0|1|1|2|2|');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1996, 991, 1996, 6,
          0.15, 0.50, 0.15, 0.08, 0);
UPDATE ITEM SET iResponse = 'SET(h1)' WHERE iID = 1996;

INSERT INTO ITEM (iID, iType, iText, iList)
          VALUES (1997, 6,     '?', '0|0|1|1|2|2|3|3|4|4|5|5|6|6|7|7|8|8|9|9|');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1997, 991, 1997, 7,
          0.30, 0.50, 0.15, 0.08, 0);
UPDATE ITEM SET iResponse = 'SET(h2)' WHERE iID = 1997;

-- colon :
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (1990, 1,   ':', 'c' );
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1990, 991, 1990, 5,
          0.45, 0.50, 0.03, 0.08, 0);
```

```
INSERT INTO ITEM (iID, iType, iText, iList)
          VALUES (1998, 6,     '?', '0|0|1|1|2|2|3|3|4|4|5|5|6|6|');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1998, 991, 1998, 8,
          0.50, 0.50, 0.15, 0.08, 0);
UPDATE ITEM SET iResponse = 'SET(m1)' WHERE iID = 1998;

INSERT INTO ITEM (iID, iType, iText, iList)
          VALUES (1999, 6,     '?', '0|0|1|1|2|2|3|3|4|4|5|5|6|6|7|7|8|8|9|9|');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
    VALUES (1999, 991, 1999, 9,
          0.65, 0.50, 0.15, 0.08, 0);
UPDATE ITEM SET iResponse = 'SET(m2)' WHERE iID = 1999;
```

6 Flagging certain patients

It's highly convenient to be able to flag certain patients as 'special' in a dynamic way. This capability should include:

1. Patients not yet seen today;
2. Patients identified (at any time) as having 'problems';
3. Patients currently flagged for 'PM review'.

Central to our strategy is the boFlag field in the BADOBS table. We clear all such BADOBS flags for a particular patient to NULL when we discharge a patient, so we are only interested in fields with a value of 0 or 1. The value 1 says we are interested in this field, and we can set it in the circumstances listed above.

First we need to be able to identify patients flagged as 'alerted'. This is trivial, given the BADOBS key value on the stack — we simply interrogate the relevant boFlag:

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(260,

  'QUERY(SELECT boFlag FROM BADOBS
    WHERE badobs = $[])->
    QOK->SKIP->FAIL->
    COPY->
    ISNULL->NOT->SKIP->FAIL->
    BOOLEAN',
'GetAlert');
```

We fail (preventing creation of any tickbox) if the value is NULL (patient discharged). Otherwise we return 1 or 0. We assume a valid key has been submitted on the stack. But how do we set the boFlag value to 1?

6.1 Patients not yet seen

First, let's look at patients 'not seen today'. This routine is invoked when we first enter the PDA program! After setting all relevant flags to 1, we will clear the boFlag in BADOBS to 0 if an EPOCH exists for a particular patient for today. Clumsy but it works: we must however examine speed issues with large numbers of observations [CHECK ME].

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (258,
'DOSQL(UPDATE BADOBS SET boFlag=1 WHERE boInactive IS NULL AND
```

```
        boFlag IS NOT NULL)->
 MARK(#0)->
 #0->
 NOW->SPLIT( )->DISCARD->
 QMANY(SELECT DISTINCT PROCESS.Person FROM EPOCH,PROCESS
  WHERE EPOCH.oMade > TIMESTAMP ''$[] 07:00:00'' AND
  EPOCH.oLength IS NOT NULL AND
  EPOCH.Process = PROCESS.process)->
 REPEAT(&UnFlagMe)->
 DISCARD',
   'FlagRecent');
```

On 2007-12-09 we altered the timestamp from 00:00:00 to 08:00:00. If somebody has been seen in the wee hours, surely they need to be seen again that day! Better is perhaps 07:00 as otherwise if the round starts a bit early, patients are flagged as 'not seen'! Another refinement is to only use epochs with an oLength value over zero.

We will use a NULL boFlag to indicate a patient who has been discharged! For each patient to be flagged we do the following:

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (259,
'COPY->BOOLEAN->SKIP->STOP->
DOSQL(UPDATE BADOBS SET boFlag=0 WHERE Person = $[]
  AND boFlag = 1)',
  'UnFlagMe');
```

The initial copy and test identifies the zero at the bottom of the stack of patients to flag. We reset the flag to zero if an observation exists for today for that patient, leaving the rest of the patients with a flag value of 1 (they *haven't* been seen).

6.2 Patients with 'a problem'

Problems are recorded as an ISPROBLEM table entry, which relates to an epoch on the relevant process. Our generic 'problem' entry is recorded related to an EPOCH on the process with ID 1, and this is the one we look for. The routine is similar to the previous one (FlagRecent), and we only look for problems flagged since midnight!

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (268,
'DOSQL(UPDATE BADOBS SET boFlag=0 WHERE boInactive IS NULL AND
        boFlag IS NOT NULL)->
 MARK(#0)->
 #0->
```

```
 NOW->SPLIT( )->DISCARD->
 QMANY(SELECT DISTINCT PROCESS.Person FROM ISPROBLEM,EPOCH,PROCESS
  WHERE ISPROBLEM.prIsOrNot = 1 AND
  ISPROBLEM.Epoch = EPOCH.epoch AND
  EPOCH.oMade > TIMESTAMP ''$[] 00:00:00'' AND
  EPOCH.Process = PROCESS.process AND
  PROCESS.ProcType = 1)->
 REPEAT(&YesFlagMe)->
 DISCARD',
   'FlagProblem');
```

Although we at present look for the 'any substantial problem' entry, we might look for 'lesser' problems by simply modifying the above routine to exclude 'AND PROCESS.ProcType = 1'. The opposite of UnFlagMe is *YesFlagMe*!

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (269,
'COPY->BOOLEAN->SKIP->STOP->
DOSQL(UPDATE BADOBS SET boFlag=1 WHERE Person = $[])',
  'YesFlagMe');
```

6.3 Patients marked for 'PM review'

```
  'QUERY(SELECT PROCESS.process FROM PROCESS
    WHERE PROCESS.Person = $[]
    AND PROCESS.ProcType = 1100
    AND PROCESS.rEnd IS NULL)->
```

Again, not dissimilar from the above but identifying extant 'PM review' processes is simpler:[91]

```
INSERT INTO FUN (fKey, fBody, fName)
   VALUES (270,
'DOSQL(UPDATE BADOBS SET boFlag=0 WHERE boInactive IS NULL AND
        boFlag IS NOT NULL)->
 MARK(#0)->
 #0->
 QMANY(SELECT DISTINCT PROCESS.Person FROM PROCESS
    WHERE PROCESS.ProcType = 1100
    AND PROCESS.rEnd IS NULL)->
 QMANY(SELECT DISTINCT PROCESS.Person FROM PROCESS
    WHERE PROCESS.ProcType = 110
    AND PROCESS.rEnd IS NULL)->
```

[91] Also more prone to abuse. Repeated changes will create numerous processes which could theoretically bring the PDA to its knees.

```
REPEAT(&YesFlagMe)->
DISCARD',
  'FlagPM');
```

On 5 December 2007 we added the code to also select all epidurals (code 110), even if they've not been 'flagged for PM review'![92]

6.4 Addendum — an epidural pop-up (930)!

Here we include a menu which we pop up the next day, after removal of an epidural.[93]

```
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (930, 20, 'Regional Check', 'RGNCHECK');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
                       miX, miY, miW, miH, miGroup)
               VALUES (930, 930, 930, 12,
                       0.000, 0.200, 0.999, 0.999, 0);
```

Here is the menu initialisation, a lot simpler than the initialisation of menu 904.

```
UPDATE ITEM SET iInitial =
  'NAME(EpR)->NAME(prR)->
    #98->#100->&ProcBetween->
     QOK->SKIP->MENU(#1)->SET(prR)->
    $[prR]->
    QUERY(SELECT ProcType FROM PROCESS WHERE process = $[])->
    QUERY(SELECT rptNature FROM PROCTYPE WHERE proctype = $[])->
    X->&FetchIdNumber->SWOP->
    TITLE($[]: $[])->
    NAME(EpL)->#1->X->&LastEpoch->SET(EpL)->
    &NewRgnEpoch->SET(EpR)'
    WHERE iID = 930;
```

We still set the title, and set up the local process variable prR as well as locating/creating a recent epoch on that process, EpR. The epoch EpL on a type 1 (general) process is needed by NewRgnEpoch as a reference point.

Here's a function to determine whether a pending check is present. The process code for a twenty-four hour check is 99. On 22/1/2008 we amended *Is24* to take a single argument, the Person in question, rather than taking this from X. This is because the transfer variable X can now be other than the person in question! All references are likewise updated.

[92] We need to look into more elegant SQL code.

[93] That is, if the regional was removed before midnight, we need to do a check on the following day!

```
INSERT INTO FUN (fKey, fBody, fName)VALUES(261,

  'QUERY(SELECT process FROM PROCESS
    WHERE ProcType = 99 AND Person = $[]
      AND rEnd IS NULL)->QOK->SKIP->RETURN(#0)->
   COPY->QUERY(SELECT rStart FROM PROCESS WHERE process = $[])->
   FLOAT->NOW->FLOAT->SWOP->SUB->FLOAT(0.95)->LESS->SKIP->RETURN->DISCARD->RETURN(
'Is24');
```

We return zero if no matching process is found. We have amended Is24 to pull out the time, and if more than the arbitrary 0.95 of the day has elapsed ('close enough'!), return the process code. The SWOP in the final line is because we always SUB the top value *from* the bottom one. If the date difference is under 0.95 days, then we skip the return statement and return zero; if greater or equal, we return the process code!

Here are the various buttons for the 24 hour check menu.

```
-- 'done' button:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (2401, 2,  'Done',  'Exitbtn', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (2401, 930, 2401, 2,
          0.750, 0.900, 0.200, 0.08, 0, 'green', 'white');
```

Here's the response to the 'Done' button:

```
UPDATE ITEM SET iResponse =
 'NOW->X->&Is24->
  DOSQL(UPDATE PROCESS SET rEnd=TIMESTAMP ''$[]'' WHERE
    process = $[])->
  POPMENU(#0)->DISCARD->DISCARD->MENU(PAINDATA)'
 WHERE iID = 2401;
```

The remaining items include an 'Ignore' button.

```
-- ignore:
INSERT INTO ITEM (iID, iType, iText, iName, iList, iLines)
          VALUES (2415, 2,  'Ignore',  'ign', '',  1);
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup, miPaper, miInk)
    VALUES (2415, 930, 2415, 2,
          0.150, 0.900, 0.200, 0.08, 0, 'red', 'white');
UPDATE ITEM SET iResponse =
 'POPMENU(#0)->DISCARD->DISCARD->MENU(PAINDATA)'
 WHERE iID = 2415;
```

```
-- weakness:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2402, 1,  'any weakness',  'prL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2402, 930, 2402, 3,
          0.05, 0.10, 0.07, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2403, 4,  'Y',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2403, 930, 2403, 3,
          0.50, 0.10, 0.07, 0.08, 1);
UPDATE ITEM SET iInitial = '"Motor"->$[EpR]->&FetchEpi->NOT' WHERE iID = 2403;
UPDATE ITEM SET iResponse =
   'SKIP->RETURN->#0->&RecordEpi(Motor)'
   WHERE iID = 2403;
-- 0 signals motor problem

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2404, 4,  'N',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2404, 930, 2404, 3,
          0.60, 0.10, 0.07, 0.08, 1);
UPDATE ITEM SET iInitial = '"Motor"->$[EpR]->&FetchEpi' WHERE iID = 2404;
UPDATE ITEM SET iResponse =
   'SKIP->RETURN->#1->&RecordEpi(Motor)'
   WHERE iID = 2404;
-- 1 signals normal motor fx!

-- numbness
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2405, 1,  'any numb areas',  'prL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2405, 930, 2405, 3,
          0.05, 0.20, 0.07, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2406, 4,  'Y',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2406, 930, 2406, 3,
          0.50, 0.20, 0.07, 0.08, 2);
UPDATE ITEM SET iInitial =
       '"Level"->$[EpR]->&FetchEpi->#11->SAME' WHERE iID = 2406;
```

```
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#11->&RecordEpi(Level)'
      WHERE iID = 2406;
-- 11 signals any sensory block
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2407, 4,  'N',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2407, 930, 2407, 3,
          0.60, 0.20, 0.07, 0.08, 2);
UPDATE ITEM SET iInitial =
       '"Level"->$[EpR]->&FetchEpi->#12->SAME' WHERE iID = 2407;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#12->&RecordEpi(Level)'
      WHERE iID = 2407;
-- 12 signals normal sensation

-- headache
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2408, 1,  'headache',  'prL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2408, 930, 2408, 3,
          0.05, 0.30, 0.07, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2409, 4,  'Y',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2409, 930, 2409, 3,
          0.50, 0.30, 0.07, 0.08, 3);
UPDATE ITEM SET iInitial =
       '"Headache"->$[EpR]->&FetchEpi' WHERE iID = 2409;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#1->&RecordEpi(Headache)'
      WHERE iID = 2409;
-- 1 signals headache
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2410, 4,  'N',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2410, 930, 2410, 3,
          0.60, 0.30, 0.07, 0.08, 3);
UPDATE ITEM SET iInitial =
       '"Headache"->$[EpR]->&FetchEpi->NOT' WHERE iID = 2410;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#0->&RecordEpi(Headache)'
      WHERE iID = 2410;
```

```
-- 0 signals NO headache

-- pain at site
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2411, 1,  'pain at site',  'prL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2411, 930, 2411, 3,
          0.05, 0.40, 0.07, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2412, 4,  'Y',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2412, 930, 2412, 3,
          0.50, 0.40, 0.07, 0.08, 4);
UPDATE ITEM SET iInitial =
       '"SitePain"->$[EpR]->&FetchEpi' WHERE iID = 2412;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#1->&RecordEpi(SitePain)'
      WHERE iID = 2412;
-- 1 signals site pain
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2413, 4,  'N',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2413, 930, 2413, 3,
          0.60, 0.40, 0.07, 0.08, 4);
UPDATE ITEM SET iInitial =
       '"SitePain"->$[EpR]->&FetchEpi->NOT' WHERE iID = 2413;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#0->&RecordEpi(SitePain)'
      WHERE iID = 2413;
-- 0 signals NO site pain


--backache:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2428, 1,  'backache',  'prL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2428, 930, 2428, 3,
          0.05, 0.50, 0.07, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2429, 4,  'Y',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
```

```
  VALUES (2429, 930, 2429, 3,
          0.50, 0.50, 0.07, 0.08, 5);
UPDATE ITEM SET iInitial =
       '"Backache"->$[EpR]->&FetchEpi' WHERE iID = 2429;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#1->&RecordEpi(Backache)'
      WHERE iID = 2429;
-- 1 signals backache
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2430, 4,  'N',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2430, 930, 2430, 3,
          0.60, 0.50, 0.07, 0.08, 5);
UPDATE ITEM SET iInitial =
       '"Backache"->$[EpR]->&FetchEpi->NOT' WHERE iID = 2430;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#0->&RecordEpi(Backache)'
      WHERE iID = 2430;
-- 0 signals NO backache


--bladder control:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2416, 1,  'incontinent',  'prL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2416, 930, 2416, 3,
          0.05, 0.60, 0.07, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2417, 4,  'Y',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2417, 930, 2417, 3,
          0.50, 0.60, 0.07, 0.08, 6);
UPDATE ITEM SET iInitial =
       '"Inconti"->$[EpR]->&FetchEpi' WHERE iID = 2417;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#1->&RecordEpi(Inconti)'
      WHERE iID = 2417;
-- 1 signals incontinence
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2418, 4,  'N',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2418, 930, 2418, 3,
          0.60, 0.60, 0.07, 0.08, 6);
```

```
UPDATE ITEM SET iInitial =
       '"Inconti"->$[EpR]->&FetchEpi->NOT' WHERE iID = 2418;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#0->&RecordEpi(Inconti)'
      WHERE iID = 2418;
-- 0 signals NO incontinence

--site check:
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2419, 1,  'site check',  'prL');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2419, 930, 2419, 3,
          0.05, 0.70, 0.07, 0.08, 0);

INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2420, 4,  'fail',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2420, 930, 2420, 3,
          0.50, 0.70, 0.12, 0.08, 7);
UPDATE ITEM SET iInitial =
       '"Site"->$[EpR]->&FetchEpi->NOT' WHERE iID = 2420;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#0->&RecordEpi(Site)'
      WHERE iID = 2420;
-- 0 signals site problem
INSERT INTO ITEM (iID, iType, iText, iName)
          VALUES (2423, 4,  'normal',  'prY');
INSERT INTO MENUITEMS (miUid, miMenu, miItem, miOrder,
          miX, miY, miW, miH, miGroup)
  VALUES (2423, 930, 2423, 3,
          0.65, 0.70, 0.20, 0.08, 7);
UPDATE ITEM SET iInitial =
       '"Site"->$[EpR]->&FetchEpi' WHERE iID = 2423;
UPDATE ITEM SET iResponse =
      'SKIP->RETURN->#1->&RecordEpi(Site)'
      WHERE iID = 2423;
-- 1 signals site OK
```

7 Process and epoch creation

We use a fairly complex set of PROCESS table entries to represent processes for their entire existence. The basic list of process types has been enumerated in the file *AnalgesiaDBpart1.tex*. Ancillary processes are contained in the *proctype.csv* file, generated either from *PerlPgm.tex* or by the creator of a particular PainForm system. Important processes with their codes include:

1 Data observation (a generic sort of 'process')

3 Patient is admitted to hospital;

5 Post-discharge planning;

99 Check of regional 24 hours after discontinuation;

500 Surgical procedure (any)

100–189 various regional *procedures*;

190 IV access

200–299 various modes of drug administration;

300–399 Modalities involving patient-controlled analgesia.

1000–1140 Ancillary process types related to organ dysfunction, chronic pain (1060), PM observation (1100), sedation (1110), blood pressure (1120), nausea (1130) and bowel opening (1140). The numbering is rather arbitrary.

Of the above, an important process is the generic 'code 1', which allows all sorts of observation, including observations describing patient particulars (NHI, Names). As important as code 1 is code 3, with which we associate bedspace allocation of a patient.[94] The other process types listed above are largely self-explanatory. A host of specialised functions serve to create such processes, e.g. *NewRgnProcInfu*.

We have defined several functions to facilitate creation of PROCESS entries:

1. *ProcAndEpoch* is used to create a code 1 process within *CreateAdmission* and *ReadmitPatient*, together with a code 3 process. ProcAndEpoch makes an *associated* epoch entry.

[94] More correct at present would be to call this 'room allocation'; although the program has the capability to represent bedspaces, we limit ourselves to rooms for now.

2. *NewProc* too creates new process entries, and is used much more widely (*OnOrOff, NewRgnProcInfu, GoIvPca, NewIvInfuProc, SetDrug, SpawnProblem*). Also see its usage in the iResponse attached to ITEM with iId 713. The NewProc procedure is similar to ProcAndEpoch, but doesn't create the associated epoch.

3. A third function is *PostDProc* which is only used in creating processes used for 'discharge planning'.

4. *DatedProc* is another more fancy version of NewProc, which permits timestamping in the past, for example recording prior insertion of an epidural.

5. *DatedPandE* is a rather clumsy variation on ProcAndEpoch, which does what DatedProc does, but throws in an epoch too. It's only used within *InsertOpData*.

Clearly there's room for rationalisation of this clumsy set of functions!

7.1 New epochs

After entering the PATIENT menu and selecting a patient, we move on to record patient details — the relevant function is *EnterDetailMenu*. This function finds the current type 1 process and attaches a new epoch to this process. This new epoch is very important for recording general data, but it also has a second purpose. When we exit the final menu for this patient (at whatever exit 'portal') we must set the oLength value to an appropriate number of milliseconds.[95] See Section 7.2 below.

A new epoch is created in many other ways. The simplest of these is to invoke *NewEpoch* (used within *EpidInfuSet, NewRgnProcInfu, IvInfuSet, NewIvDrug,* and *SpawnProblem*). A dressed-up version of NewEpoch is *FancyEpoch*, which in addition to producing a new observation, leaves the existing process code on the stack (Ugh)! FancyEpoch is invoked from within *ForceEpoch*, which is strangely enough only used in comment creation. Comments are generally attached to an epoch of a type 1 process. (See *MakeGeneralComment*).

NewRgnEpoch is a rather clumsy epoch-creator, only used in the setup of menus with item IDs 904 and 930 (See usage).

[95] Which we do by subtracting the epoch oMade value from the then current time, and converting into milliseconds.

7.2 End of an epoch

The *EndEpoch* function is invoked wherever we terminate an epoch. This function enters a duration value in the oLength field of the relevant entry in the EPOCH table. The duration is in milliseconds (although effectively we record seconds, as we're using timestamps accurate to the nearest second). The points at which epoch termination can occur are:

- Within the FINISH menu;
- When a patient is discharged, whether they died (*PatientDied*) or were normally discharged (*DoDischarge*);
- When we return abnormally from the first patient data menu (abort the session within the DETAILS menu, code 903).

Clicking on the [Done] button in the FINISH menu (code 909, see Section 5.16) terminates this patient encounter. Abnormal return from the first patient data menu (DETAILS) occurs when we click on the button with an iID of 132.

7.3 Conventions for epoch variables

It's convenient to store certain special epochs in local variables. Important in this regard are:

EpQ This epoch-related local variable, accessed using `->$[EpQ]->`, contains the most recent epoch associated with an infusion process (in an appropriate menu). The process is customarily stored in the local variable `prQ`. The process might be a straightforward infusion (codes 200–299), or involve patient-controlled analgesia (process codes 300–399).

EpR Used for the current epoch, which is attached to a process (prR) that describes a regional procedure. R is for Regional.

EpL Used for a general observational epoch, associated with the current general observational process. L is for Latest.

8 Menu hierarchies and caching

This entire section refers only to optimisation of caching on the *PDA*. On the desktop, we assume (at present) that the SQL is optimised and fast enough. The fundamental caching principle is that the caching should in no way affect the logical functioning of our database, in other words, no modification is required to SQL code when caching is turned on!

We've structured our language to limit the direct transmission of information between menus to a single number (the X-variable). All other flow of information is through the database. This simple approach encourages loose coupling between components, and also suggests that if we cache database information appropriately, we can vastly improve performance. The latter turns out to be correct — by simply caching relevant rows from the PROCESS and EPOCH tables,we dramatically improve SQL performance, particularly with large database files.

In order to be able to cache correctly, we need to know where particular database information is required. For example, if a series of menus relates solely to a particular patient (as is mostly the case) then all other patient information can be excluded by using the CACHE command correctly. When we exit this state, then we can UNCACHE the relevant database tables.

8.1 A note on 'static' data

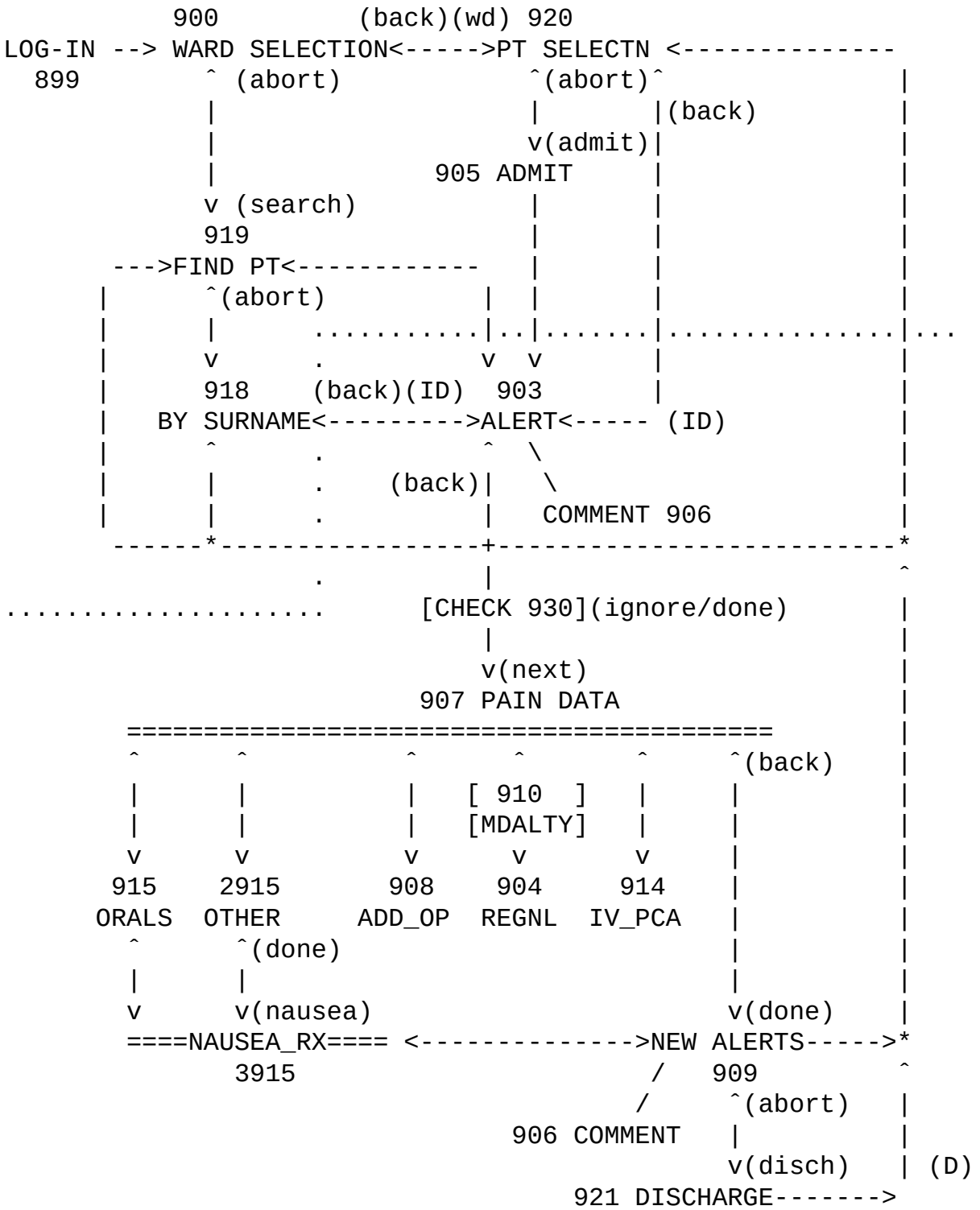
On the PDA, the data which define all menus and their components (contained in the tables ITEM, MENUITEMS and ICOLTABLE) should not often (at present, not ever) be altered on the PDA. This suggests several possible caching approaches:

1. Checking an exact SQL query string, and caching the value(s);
2. Caching depending on the particular menu being used (with or without the preceding approach);
3. More baroque caching/coding rules (unwise).

We haven't yet implemented the above approaches.

8.2 Menu hierarchy

The hierarchy of menus (and functions which move between them) is as follows. (Ultimately we will turn this sketch into a real diagram). Our initial caching demarcated the ward and patient selection (and patient searches) from all of the other menus, which deal with a specific patient, indicated by the dotted line in the following diagram.


900 (back)(wd) 920
LOG-IN --> WARD SELECTION<----->PT SELECTN <--------------
899 ^ (abort) ^(abort)^ |
	(back)
v(admit)	
905 ADMIT	
v (search)		
919		
--->FIND PT<------------		
^(abort)				
	.....|..|.....|.....|...
v . v v	
918 (back)(ID) 903	
BY SURNAME<--------->ALERT<----- (ID)
^ . ^ \
	. (back)| \
	.	COMMENT 906
------*-----------------+-------------------------*
. | ^
..... [CHECK 930](ignore/done) |
| |
v(next) |
907 PAIN DATA |
========== |
^ ^ ^ ^ ^ ^(back) |
| | | [910] | | |
| | | [MDALTY] | | |
v v v v v | |
915 2915 908 904 914 | |
ORALS OTHER ADD_OP REGNL IV_PCA | |
^ ^(done) | |
| | | |
v v(nausea) v(done) |
====NAUSEA_RX==== <--------------->NEW ALERTS----->*
3915 / 909 ^
/ ^(abort) |
906 COMMENT | |
v(disch) | (D)
921 DISCHARGE------>

8.3 Logging in

See Section 5.21.

Referred by

- (Nobody) — only invoked on entering main program!

Connects to

- Ward selection (8.4).

Functions used

- FlagRecent
- ListUsers (complex join on PERSDATA, EPOCH, PROCESS and PERSON)
- ProcessUserNames
- Retrieve

Tables used

- PERSDATA
- EPOCH
- PROCESS
- PERSON

Caching opportunities

Major delay is the massive join referred to above. The size of the PROCESS and EPOCH files is big.

- If we could limit PERSON.pStatus *over 1*, and then based on this propagate through PERSON:PROCESS.Person and thence to PROCESS:EPOCH.Process, we might dramatically lower search times.[96]

[96] But . . . at present we have no ability to check for anything other than a single value match in an Int32 field, so we would need to modify PDA routines to deal with pStatus field.

8.4 Ward selection

See Section 5.1

Referred by

- Logging in (8.3).
- Back from Find patient (8.7)
- Back from Patient selection within ward (8.5)

Connects to

- Find patient (Search: 8.7)
- Patient selection within ward (8.5) — by clicking on a ward name
- (Quit) option is important.

Functions used

- ListWards

Tables used

- WARD (all components must be accessible)

Caching opportunities

These are minimal at present.

- If we later decide to highlight certain wards based on occupancy, then we will also need to access BADOBS. It's possible that caching might improve speed, but unlikely (How many times will the table be accessed??). Under such circumstances, we might cache on BADOBS.boFlag being set to 1.

8.5 Patient selection

See Section 5.2

Referred by

- Ward selection 8.4
- Admit (8.6)
- Alert (8.9)
- New alerts (8.22)
- Discharge (8.23)

Connects to

- Admit (8.6)
- Alert (8.9)

Functions used

- ListRooms (ROOM)
- ManyBack
- GetBadObs4Ward (BADOBS)
- FetchSurname (PERSDATA)
- FetchIdNumber (PERSDATA)
- EnterDetailMenu[97]
- GetAlert (BADOBS)
- GetPatientRoom (BADOBS)
- SetNewRoom (BADOBS, PROCESS, EPOCH)

[97] Note here: ReadmitPatient was a potential problem, as caching became deranged due to an error in placement of the CACHE statements. Fixed!

Tables used

- ROOM
- BADOBS
- PERSDATA
- PROCESS
- EPOCH

Caching opportunities

We already cache once we *leave* this menu to enter the alerts for a particular patient. Other opportunities include:

- ROOM:Ward is the particular ward value. Might speed things a bit, but note BADOBS is already de-normalised to speed up fetch.
- BADOBS:boInactive being reset would limit the field, but as we only perform a major SELECT once, perhaps not that useful. In the light of all of the queries of BADOBS, still perhaps an option.
- It is tempting (on entering this menu) to cache on BADOBS.Bed.{range} where bed values are limited to this ward[98]. We cannot however then extend this caching to EPOCH (or thence to PROCESS) as we would be eliminating other vital observations on particular patients.[99]

8.6 Patient admission

See Section 5.3

Referred by

- Patient selection(8.5).

[98] Again requiring that we extend caching to a *range*!

[99] If however we were to extend the functionality of our caching to permit storage of a value other than the primary key of a row, then we might store the Person of BADOBS, and then limit via PROCESS and EPOCH! Something along the lines of CACHE(BADOBS.Bed.{40000–49999}@Person) followed by CACHE(BADOBS:PROCESS.Person) and CACHE(PROCESS:EPOCH.Process). Even more complex logic would permit limiting by boInactive.

Connects to

- Patient selection(8.5) on abort.
- Alerts (8.9).

Functions used

- EnterDetailMenu
- DoWholeAdmission
- AdmitPatient (PERSON)
- CreateAdmission (BADOBS)
- ProcAndEpoch (PROCESS, EPOCH)
- KeepPersonData (PERSDATA)
- RecordASA (MEDSCORE)
- RecordWeight (MEASURE)
- RecordDob (PERSON)

Tables used

- PERSDATA (including check on hospital number)
- PERSON
- BADOBS
- PROCESS
- PERSDATA
- MEDSCORE
- MEASURE
- EPOCH

Caching opportunities

There seems to be little opportunity for useful caching in this menu.

8.7 Find a patient

See Section 5.4.

Referred by

- Ward selection 8.4
- Alerts (8.9);
- Selection by surname (8.8)
- New alerts (8.22)
- Discharge (8.23)

Connects to

- Alerts (8.9);
- Selection by surname (8.8).
- Ward selection 8.4

Functions used

- EnterDetailMenu (PROCESS, EPOCH)
- GoSurname

Tables used

- PERSDATA (in searching on hospital number, also for GoSurname)
- PROCESS, EPOCH (of little utility in caching)

Caching opportunities

Limited.

8.8 Selection from a list of surnames

See Section 5.5.

Referred by

- Find patient (8.7)
- New alerts (8.22)
- Discharge (8.23)

Connects to

- Alerts (8.9);
- Find patient (8.7)

Functions used

A lot of the functionality is similar to selection of a patient from a ward (Section 8.5). The exceptions are SearchBySurname, and GetPatientWard which resembles GetPatientRoom.

- SearchBySurname (PERSDATA, EPOCH, PROCESS, PERSON)
- GetPatientWard
- FetchIdNumber
- EnterDetailMenu
- FetchSurname

Tables used

- PERSDATA, EPOCH, PROCESS
- PERSON

Caching opportunities

Despite the big join in SearchBySurname, the function is fairly fast and probably doesn't need any caching fiddles in its current form.

8.9 Patient alert screen

See Section 5.6. This menu and several subsequent menus are conveniently cached based on the patient selected, limiting both PROCESS and EPOCH dramatically.

Referred by

- Patient selection within ward (8.5)
- Find patient (Search: 8.7)
- Selection by surname (8.8)
- Comment (8.10)
- Pain data (8.12)
- Admit (8.6)

Connects to

All of the above, apart from Admit.

- Patient selection within ward (8.5)
- Find patient (Search: 8.7)
- Selection by surname (8.8)
- Comment (8.10)
- Pain data (8.12)

Functions used

- FetchIdNumber (PERSDATA)
- FetchSurname (PERSDATA)
- Is24
- FetchASA
- FetchMedScore (EPOCH, PROCESS, MEDSCORE)
- FetchForename (PERSDATA)
- ListWards
- FetchWeight
- FetchAge

- WasItOn (PROCESS)
- OnOrOff
- GetMyWard (BADOBS, WARD)
- SetNewWard (PROCESS, EPOCH, BADOBS)

Tables used

- PROCESS
- EPOCH
- COMMENT (directly apart from above functions)
- MEDSCORE
- PERSDATA
- BADOBS
- WARD

Caching opportunities

The current caching seems to have sorted out most of the problems. It is tempting to examine the COMMENT table in more detail. We must check this out with a vast number of comments (from IDAS)!

8.10 Comments

See Section 5.7

Referred by

- Alerts (8.9);
- New Alerts (8.22).

Connects to

- Alerts (8.9);
- New Alerts (8.22).

Functions used

- FetchIdNumber
- FetchSurname
- MakeGeneralComment
- FindRecentProcess
- ForceEpoch
- FancyEpoch
- ShortDate

Tables used

- COMMENT
- EPOCH
- PROCESS

Caching opportunities

Although we might conceivably need to tune this menu, simply caching on the individual patient is probably sufficient (as already performed in menus leading to the comments menu).

8.11 Epidural pop-up

See Section 6.4 and the comment here.

Referred by

- Alerts (8.9);

Connects to

- Pain data (8.12)

Functions used

- FindRegional
- FetchIdNumber
- LastEpoch
- NewRgnEpoch
- Is24
- FetchEpi
- RecordEpi

Tables used

As usual for the above functions. (Perhaps fill in).

Caching opportunities

Limited once we've cached PROCESS and EPOCH on the patient.

8.12 Pain data

See Section 5.8

Referred by

- Alerts (8.9);
- New alerts (8.22)
- Pain help (8.13)
- Add an operation (8.14)
- Regionals (8.15)
- IV PCA (8.17)
- Oral Rx (8.19)
- Other modalities (8.21)
- Epidural pop-up (8.11)

Connects to

All of the referrers above, with the exception of the epidural pop-up.

Functions used

- LastEpoch
- FetchIdNumber
- FetchSurname
- PainGet, SetPain, ReplaceNull, FindPainScore, NewPainScore
- SpawnProblem, SetProblem, FetchProblem
- CheckIsEvent, CheckNonevent
- ProcBetween
- JustKillregional
- FailAndReload
- KillManyProcs
- KillProc
- JustKillOther
- JustKillOrals
- NewProc

Tables used

- EPOCH,PROCESS
- PERSDATA
- PAINSCORE
- ISPROBLEM
- NONEVENT

Caching opportunities

Most important is caching the individual (already done). Although we might potentially cache at several other points, we would probably derive most benefit from caching on the menu components themselves rather than user data tables.

8.13 Pain help

See Section 5.18.1. Only entered on clicking on the header of the pain information menu.

Referred by

- Pain data (8.12)

Connects to

- Pain data (8.12)

Functions used

- ShortDate
- FindAdministration
-
-

Tables used

- PAINSCORE, EPOCH, PROCESS
- PCA

Caching opportunities

Minimal?

8.14 Add an operation

See Section 5.9

Referred by

- Pain data (8.12)

Connects to

- Pain data (8.12)

Functions used

- ShortDate
- FetchIdNumber
- FetchSurname
- InsertOpData
- ListOpTypes
- DatedPandE

Tables used

- EPOCH
- PROCESS
- SURGTYPE, SURGETYPEOB

Caching opportunities

Minimal?

8.15 Regionals

See Section 5.10. This section is *very* complex.

Referred by

- Pain data (8.12)
- Regional help (8.16)

Connects to

As for referrals.

Functions used

- LastEpoch
- SetEvent
- FindRegional
- FetchIdNumber
- NewRgnEpoch
- CheckInfusion
- FindInfuObs
- ProcBetween
- RecentProcobs
- RecordEpi
- FetchEpi
- PcaRecord
- pcAble
- GetPca
- FindTopups
- InfuRateSet
- SetTotal
- FindTotal
- ListDrugs
- GetInfusionLabel
- EpidInfuSet

- FindAdministration
- FailAndReload
- NewRgnProcInfu
- ChangeEpidInfusion
- NewEpoch
- KillProc
- TopupSet
- NewRxObs
- NewPca
- IsItPcra
- TogglePcea
- StopInfusion

Tables used

- RX
- PROCESS, EPOCH
- RGNOBS, INFUSIONOBS, RXOBS

Caching opportunities

We need to explore these in detail as the menu can be a little sluggish, not surprising in view of the complexity of the database access. We might do something with the RGNOBS, RXOBS or INFUSIONOBS tables, depending on profiling. We should also look into caching ‘static’ menu elements.

8.16 Regional help menu

See Section 5.18.3.

Referred by

- Regionals (5.10)

Connects to

- Regionals (5.10)

Functions used

- FindRegional (ProcBetween)
- FetchEpi
- ShortDate

Tables used

- PROCESS
- RGNOBS

Caching opportunities

Need exploration.

8.17 IV PCA

See Section 5.11. Resembles the above regional menu (8.15).

Referred by

- Pain data (8.12)
- PCA help (8.18)

Connects to

- Pain data (8.12)
- PCA help (8.18)

Functions used

- JustKillPca
- SetNonevent
- LastEpoch
- FindAdministration
- FetchIdNumber
- FetchSurname
- FailAndReload
- KillManyProcs
- CheckNonevent
- NewProc
- GoIvPca
- PcaRecord
- GetPca
- SetTotal
- FindTotal
- InfuRateSeet
- FindInfuRate
- GetInfusionLabel
- IvInfuSet
- PcaNoteSettings
- GetPcaSet
- NewIvDrug
- ChangeIvInfusion
- NewEpoch

Tables used

- PROCESS
- RX
- PCASETTINGS
- RXOBS
- PCA

Caching opportunities

Unclear.

8.18 PCA help menu

See Section 5.18.2.

Referred by

- IV PCA (8.17)

Connects to

- IV PCA (8.17)

Functions used

- FindAdministration
- ShortDate

Tables used

- EPOCH
- PCA

Caching opportunities

Needs research.

8.19 Oral therapy

See Section 5.13

Referred by

- Pain data (8.12)
- Nausea Rx (8.20)

Connects to

- Pain data (8.12)
- Nausea Rx (8.20)

Functions used

- JustKillOrals
- LastEpoch
- SetEvent
- FetchIdNumber
- FetchSurname
- CheckNonevent
- FindAdministration
- SetNonevent
- KillManyProcs
- GetDrugProcs
- GetTradeName
- AskStopDrug
- FancyEpoch
- NewRxObs2
- Set24

- Get24hr
- ListDrugs
- SetDrug
- ByFormulation

Tables used

- RX
- DRUG
- EPOCH
- RXOBS

Caching opportunities

Need clarification. Requirement is unclear.

8.20 Nausea Rx

See Section 5.14.

Referred by

- New alerts (8.22)
- Other modalities (8.21)
- Oral therapy (8.19)

Connects to

- New alerts (8.22)
- Other modalities (8.21)
- Oral therapy (8.19)

Functions used

- LastEpoch
- FetchIdNumber, FetchSurname
- GetNauseaRxProcs
- GetTradename
- AskStopDrug
- Set24
- Get24hr
- SetDrug

Tables used

- PROCESS
- RX

Caching opportunities

Unclear.

8.21 Other modalities

See Section 5.15.

Referred by/connects to

- Pain data (8.12)

Functions used

- LastEpoch
- FetchIdNumber, FetchSurname
- CheckNonevent
- FindAdministration

- GoOther
- Get24hr
- Set24
- GetDrugProcs
- AskStopDrug
- GetTradeName
- ListDrugs
- SetDrug
- ByFormulation

Tables used

- PROCESS, EPOCH
- RX, RXOBS

Caching opportunities

To clarify?

8.22 New Alerts

See Section 5.16.

Referred by

- Pain data (8.12)
- Nausea Rx (8.20)
- Discharge (8.23)

Connects to

- Pain data (8.12)
- Nausea Rx (8.20)
- Discharge (8.23)
- Patient selection (8.5)
- Find patient (8.7)

Functions used

- LastEpoch, FetchIdNumber, FetchSurname
- UnFlagMe
- PatientDied
- FetchProblem, SpawnProblem, SetProblem
- FindRecentProcess, NewProc, NewEpoch, NewProblem
- EndProcByType

Tables used

- PERSON, EPOCH, PROCESS
- BADOBS
- ISPROBLEM

Caching opportunities

Needs work.

8.23 Discharge

See Section 5.17.

Referred by

- new alerts (8.22)

Connects to

- new alerts (8.22)
- Patient selection (8.5)
- Find patient (8.7)

Functions used

- FetchIdNumber, FetchSurname
- DoDischarge
- GetDProc
- PostDProc
- KillManyProcs

Tables used

- PROCESS

Caching opportunities

Caching is unlikely to help.

9 PDA scripting conveniences

In order to decrease the burden of having long scripts or convoluted PDA code (and because the scripting language is so convenient) we exported a lot of functionality to scripts. We've slightly clunkily included these scripts in the *MENUS.sql* file. Here they are:

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (121, 'QUERY(SELECT ITEM.iID FROM ITEM WHERE
                 ITEM.iName = ''$[]'')',
  'cSQ1');

INSERT INTO FUN (fKey, fBody, fName)
  VALUES (122, 'QUERY(SELECT
     MENUITEMS.miX,MENUITEMS.miY,MENUITEMS.miW,MENUITEMS.miH
     FROM MENUITEMS WHERE MENUITEMS.miItem = $[])',
  'cSQ2');

INSERT INTO FUN (fKey, fBody, fName)VALUES(123,
  'QUERY(SELECT ITEM.iText FROM ITEM WHERE ITEM.iID = $[])',
  'cSQ3');

INSERT INTO FUN (fKey, fBody, fName)
  VALUES (124, 'QUERY(SELECT ITEM.iInitial,ITEM.iText,ITEM.iType
    FROM ITEM WHERE ITEM.iID = $[])',
  'cSQ4');

INSERT INTO FUN (fKey, fBody, fName)
  VALUES (125, 'QUERY(SELECT ICOLTABLE.irItem,ICOLTABLE.irFraction
    FROM ICOLTABLE WHERE ICOLTABLE.irTBL = $[]
      AND ICOLTABLE.irOrder = 1)',
  'cSQ5');

INSERT INTO FUN (fKey, fBody, fName)
  VALUES (126, 'QUERY(SELECT ITEM.iType,ITEM.iInitial FROM ITEM
     WHERE ITEM.iID = $[])',
  'cSQ6');
```

The following SELECT statement is rather important. It selects all menu items (the relevant properties) for a given menu.

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (127, 'QMANY(SELECT miEnabled,miGroup,
  miX,miY,miW,miH,miItem FROM MENUITEMS WHERE miMenu = $[]
  ORDER BY MENUITEMS.miOrder DESC)',
  'cSQ7');
```

```
INSERT INTO FUN (fKey, fBody, fName)
  VALUES (131, 'QUERY(SELECT ITEM.iInitial FROM ITEM
    WHERE ITEM.iID = $[])->RUN',
  'cSQ8');

INSERT INTO FUN (fKey, fBody, fName)
  VALUES (132, 'QUERY(SELECT ITEM.iLines FROM ITEM
    WHERE ITEM.iID = $[])',
  'cSQ9');

INSERT INTO FUN (fKey, fBody, fName)
  VALUES (133, 'QUERY(SELECT ITEM.iResponse FROM ITEM
    WHERE ITEM.iID = $[])',
  'cSQ10');

INSERT INTO FUN (fKey, fBody, fName)
  VALUES (136,
  'QMANY(SELECT irItem,irName,irEnabled,irFraction
    FROM ICOLTABLE WHERE irTBL = $[] ORDER BY irOrder DESC)',
  'cSQ11');

INSERT INTO FUN (fKey, fBody, fName)
  VALUES (140, 'QUERY(SELECT ITEM.iInitial FROM ITEM
     WHERE ITEM.iID = $[])->RUN',
  'cSQ12');

INSERT INTO FUN (fKey, fBody, fName)
   VALUES (141, 'COPY->QUERY(SELECT UIDS.u$[]
     FROM UIDS WHERE UIDS.uKey = 1)->
   COPY->BURY->#1->ADD->
   DOSQL(UPDATE UIDS SET UIDS.u$[]=$[]
         WHERE UIDS.uKey = 1)->DIGUP',
   'NEWKEY');


*
-- this signals the end of the file!
```

The NEWKEY function is really odd, as we use it to generate new key values for arbitrary columns, something that we should really make atomic, and write as C++ code! (The Perl already has this functionality).

10 Addendum: specific stuff

For a particular institution, you will need to insert an enormous amount of very specific data. Although we might leave you to your own devices, I thought it prudent to provide an example of what you're up against. Here follow tentative data for my institution, written to the file *specific.sql*.

10.1 Ward specifics

```
-- population of WARD table:

INSERT INTO WARD (ward, swrdText)VALUES(1, 'new');
```

In the following, the room with code 100 corresponds to a generic room in the 'unlisted' ward!

```
-- population of ROOM table etc:

INSERT INTO ROOM (room, Ward, srmText)VALUES(100, 1, '?');

INSERT INTO BED (bed, Room, sName)VALUES(10000, 100, '?');
```

10.2 Drug-specific data

10.2.1 A note on drug codes

Different countries have different codes. For example, in the USA, the FDA mandates a ten digit 'National Drug Code' or NDC.[100] The NDC is often used in HIPAA (another US set of regulations) with a leading zero in one of the fields, causing much confusion! Here's the blurb![101] Also from the USA, the venerable AMA DRUG Evaluation Subscription classification scheme hasn't been changed since 1976! On packages in the USA, the barcode begins with 3 or 03 (UPC-A and EAN-13 respectively) and there's a check-digit at the end. The UPC or universal product code dates back to 1973! There are also HCPCS codes,

On the international scene, we have the WHO "International Nonproprietary Names" (http://www.who.int/medicines/services/inn/en/). The INN may differ from the US Adopted Name but should be the same as the British Approved Name (defined in the British Pharmacopoeia). The BAN is useful in that unique generic names are assigned to drug combinations! There should now be harmonisation between the BP and the European Pharmacopoeia.

We also have the WHO's "ATC codes", which provide a hierarchical classification of drugs by class. ATC stands for 'Anatomical Therapeutic Chemical', and is a hierarchical system which classifies substance according to organ/system affected, as well as pharmacological, chemical and therapeutic properties. A term

[100] This comprises a 4 or 5-digit labeller code which describes the drug 'manufacturer', followed by a product code and a package code. The length of each of the 3 'segments' varies depending on the FDA and the company.

[101] http://www.fda.gov/cder/ndc/

commonly associated with ATC is DDD, for 'Defined Daily Dose'. ATC represents an extension of the older European Pharmaceutical Market Research Association classification system (EphMRA). ATC has five levels, the first containing 14 main groups (A–D,G,H, J, L, M, N, P, R, S, V), and things degenerating into chaos thereafter. A drug has only one place in this system as it is regarded as having a main active ingredient and a main therapeutic use. Where there is controversy, a WHO committee decides!

A listing of codes is not readily found on the 'net, but try: http://www.msupply.org.nz/ or Google ["ATC system" caries belladonna].

10.2.2 Epidural infusions

This needs further shaping; we have recently (5/2007) moved population of the DRUG table into the CSV import section of *PerlPgm.tex*.

```
INSERT INTO DRUGFORM (drugform, dformText)
  VALUES (1, 'Epidural');
```

10.2.3 Intravenous PCA

Exported to CSV, similar to the above. Rather artificial.

```
INSERT INTO DRUGFORM (drugform, dformText)
  VALUES (8, 'IV PCA');
```

10.2.4 Other IV, non-PCA, and SC

```
INSERT INTO DRUGFORM (drugform, dformText)
  VALUES (4, 'IV');
```

The following (for subcutaneous Rx) is a bit of a hack:

```
INSERT INTO DRUGFORM (drugform, dformText)
  VALUES (9, 'SC');
```

10.2.5 Orals

```
INSERT INTO DRUGFORM (drugform, dformText)
  VALUES (10, 'Tabs'),
         (17, 'Syrup'),
         (13, 'Caps');
```

10.2.6 Rectal (PR) drugs

```
INSERT INTO DRUGFORM (drugform, dformText)
  VALUES (20, 'Suppository');
```

10.2.7 Transdermal drugs

```
INSERT INTO DRUGFORM (drugform, dformText)
  VALUES (30, 'Patch');
```

10.2.8 Special infusions

From above, DRUGFORM code 8 is an intravenous PCA infusion. See the CSV imports, as noted above.

10.3 Test data for patient selection

To allow testing of patient selection from a ward, we formerly rather artificially populated certain tables using SQL statements. We now do so by importing CSV files (See *PerlPgm.tex*) and have removed the SQL previously residing here!

```
*
-- this signals the end of the file!
```

The above lines terminate the file and we shouldn't try to add more SQL subsequent to these lines!

10.4 *A list of all functions*

Finally, here's a table containing links to all functions, for convenience (this still needs to be sorted in order, and a few functions are missing).

ListWards	EnterDetailMenu	GetBadobs4Ward
-	Retrieve	FetchSurname
FetchIdNumber	GetPatientRoom	SetNewRoom
GetPmFlag	DoWholeAdmission	AdmitPatient
CreateAdmission	ProcAndEpoch	KeepPersonData
RecordASA	RecordASA	LastEpoch
GetMyWard	SetNewWard	FetchForename
FetchMedscore	FetchASA	FindRecentProcess
WasItOn	WasBetween	EndProcByType
NewProc	OnOrOff	MakeGeneralComment
ForceEpoch	FancyEpoch	ShortDate
ListConsultants	NoteConsultant	FetchConsultant
SetPain	FindPainScore	NewPainScore
PainGet	ReplaceNull	CheckNonevent
SetNonevent	Fail	ListOpTypes
-	InsertOpData	JustKillRegional
ProcBetween	FindRegional	RecentProcObs
NewRgnEpoch	KillProc	FetchEpi
RecordEpi		FindAdministration
FindInfuObs	FindInfuRate	InfuRateSet
EpidInfuSet	NewRgnProcInfu	ChangeEpidInfusion
NewEpoch	NewProc	GetInfusionLabel
FailAndReload	TopupSet	FindTopups
NewRxObs	PcaRecord	GetPca
NewPca	Able	IsItPcra
TogglePcea	KillManyProcs	StopInfusion
GoIvPca	IvInfuSet	NewIvDrug
ChangeIvInfusion	PcaNoteSettings	NewPcaSet
GetPcaSet	FindTotal	SetTotal
JustKillPca	JustKillOrals	GoOrals
GetDrugProcs	ListDrugs	SetDrug
NewRxObs2	SpawnProblem	NewProblem
FetchProblem	SetProblem	DoDischarge
GoOrals	GoOther	JustKillOther
GetTradeName	Get24hr	Set24
AskStopDrug	SearchBySurname	GetPatientWard

DatedPandE	ReadmitPatient	CheckIsEvent
NewNonevent	SetEvent	ListUsers
GetRegionalModes	CheckInfusion	ListRooms
RecordWeight	FetchWeight	GetNauseaRxProcs
PatientDied	GetDProc	PostDProc
FlagRecent	GetAlert	UnFlagMe
Is24	-	RecordDob
FetchAge		

Click on one of the above links to visit the function.

11 Change Log

From version 0.95, we introduce a change log.

11.1 Version 0.95

1. We've altered the two invocations of SetNewWard, so that moving the patient to a new ward is confirmed. (Ideally this change should be moved to within SetNewWard, but we haven't yet done so).

2. On 31/3/2008 we introduced the error handling menu. There are several possible ways of doing this:

   (a) Attach the PAINERROR table directly to a process (e.g. the offending process, or the observation process)

   (b) Attach this table to the relevant EPOCH (or create one)

   (c) Create a separate type of PROCESS for errors, and attach the table either directly, or via an EPOCH.

   Advantages of the last approach are the minor one that all patients with error processes can easily be identified (but a query linking back from the PAINERROR table isn't much of a hassle), and that we are 'isolating' the errors from other processes, retaining error-specific processes and epochs. This approach seems cleaner, and does perhaps allow us to more generally describe errors — If we are 'in the epidural menu' but identify an error, we might not wish to associate the error identified with the epidural process. The last mentioned 'benefit' might also be considered a liability in that we are losing some structure — we associate the error with the error process and *not* directly with the offending process, only making this association by implication. This last argument is a pretty potent one for not having a separate error process, so we'll discard the third option.

   Should we attach the PAINERROR table directly to a PROCESS, or indirectly via an EPOCH? The latter does give us a finer structure, and means that we don't have to add a timestamp to each PAINERROR table entry. It allows us to associate several errors with a particular EPOCH, and the errors with a given period of observation (but only if this already exists), a fairly good argument for the EPOCH approach.

   There are two residual problems:

   (a) We need a mechanism for generally identifying errors not associated with one of the four 'modalities' or 'modality groups' (regional, IV

PCA, enteral and ‘other’) — this might involve attaching an error through an epoch on the current general observation process, an attractive option;

(b) With the current structure of our enteral and ‘other’ menus, we cannot easily associate the error table directly with a given drug! Options here might be to:

- Alter the menu to have an error button next to each item (cumbersome and undesirable)
- Alter the response to clicking on a drug (either having two questions, one of them “Stop the drug?” and the other “Report error”; or opening up a sub-menu, surely both unacceptable in terms of ergonomics)
- Have a more complex error menu for these choices, where we pull out the drugs (processes) and then permit a single click on a drug to indicate that ‘this was the drug/process involved.
- Not go down to the drug level, here simply attaching the record of the error to a more generic observation process, a somewhat unattractive but not totally excluded option.

My ‘solution’ is as follows:

(a) The main error menu lists candidate processes (with their date, type, and associated drug, if any)

(b) The user clicks on a table line which represents one of these processes

(c) A subsidiary menu pops up permitting attachment of a PAINERROR entry to the most recent observation on that process.

As a minimum we introduce an [Err] button in four menus, related to regionals, PCA, orals and ‘other’, based on the creation of a PAINERROR menu in *AnalgesiaDBpart1.tex*. There are several things we need to do apart from creating and inserting the new menu into the database, and the relevant button into the menus. These include:

(a) Inserting the associated FUN table entries;

(b) Ensuring the relevant PAINERROR key generator is inserted into UIDS, and the xTABLE and xCOLUMN entries are updated manually!

```
ALTER TABLE UIDS ADD uPainerror integer;
UPDATE UIDS SET uPainerror = 1000;
-- next fix xtable/xcolumn entries:
```

```
 select xtakey FROM XTABLE WHERE xTaname = 'EPOCH'
 -- gives eg 103
 select uxtable, uxcolumn, uxlimit from uids;
 --- this gives eg {134, 391, 373}
INSERT INTO xTABLE (xTaKey, xTaName) VALUES (134, 'ERRTYPE')
INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize, xCoTa
               VALUES (391, 'cold',  'I',      4,       134)
INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize, xCoTa
               VALUES (392, 'errtype','I',     4,       134)
INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize,xCoTab
               VALUES (393, 'etText', 'V',     32,      134)
INSERT INTO xTABLE (xTaKey, xTaName) VALUES (135, 'PAINERROR
INSERT INTO xLIMIT (xLiKey, xLiName, xLiType, xLiColumn)
              VALUES (373,  'baderrtype','P', 392);

    constraint baderrtype primary key (errtype),

INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize, xCoTa
               VALUES (394, 'cold',  'I',      4,       135)
INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize, xCoTa
               VALUES (395, 'painerror','I',   4,       135)
INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize,xCoTab
               VALUES (396, 'Epoch', 'I',      4,       135)
INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize, xCoTa
               VALUES (397, 'Errtype','I',     4,       135)
INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize,xCoTab
               VALUES (398, 'ErrStamp','S',    14,      135)
INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize,xCoTab
               VALUES (399, 'peText', 'V',     32,      135)
INSERT INTO xLIMIT (xLiKey, xLiName, xLiType, xLiColumn)
              VALUES (374,  'badpnerr','P',   395);
INSERT INTO xLIMIT (xLiKey, xLiName, xLiType, xLiColumn, xLi
              VALUES (375,    'badPEepoch','F',396 ,103 );
INSERT INTO xLIMIT (xLiKey, xLiName, xLiType, xLiColumn, xLi
              VALUES (376,    'badPEtype','F',397 ,134 );
--- we must also set a new xCOLUMN value for uPainerror (sub
INSERT INTO xCOLUMN (xCoKey, xCoName, xCoType, xCoSize,xCoTab
               VALUES (400, 'uPainerror', 'I', 4,      4);

UPDATE UIDS SET uxTable = 136, uxColumn = 401, uxLimit = 377
```

(c) Ensuring the tables are exported to the PDA (this is largely related to the x-tables noted above)

(d) Inserting the table names ERRTYPE and PAINERROR into *XLOG.LST*

(e) Ensuring that on making the associated patients ‘cold’, these tables are also affected! [EXPLORE]

(Note how the PDA silently failed when we had two spaces thus “DOSQL(INSERT INTO PAINERROR(” in ReportError. [EXPLORE LATER])

(f) In the epidural menu (904) we must also move the ‘low BP’ text and buttons up and out of the way! [AND FIX SQL ERROR which occurred with new prescription!?]

```
UPDATE MENUITEMS SET miX = 0.75, miY = 0.68 WHERE miUid = 432
UPDATE MENUITEMS SET miX = 0.85, miY = 0.68 WHERE miUid = 433
UPDATE MENUITEMS SET miX = 0.60, miY = 0.68 WHERE miUid = 434
```

(g) Some PROCTYPE entries WERE a tad long-winded, for example ‘enteral drug administration’. We must shorten this to ‘Enteral drug’. Here’s the big list . . .

```
UPDATE PROCTYPE SET rptNature ='Enteral drug' WHERE PROCTYPE
UPDATE PROCTYPE SET rptNature ='Epidural catheter' WHERE PRO
UPDATE PROCTYPE SET rptNature ='Epidural drug' WHERE PROCTYP
UPDATE PROCTYPE SET rptNature ='Epidural PCEA' WHERE PROCTYP
UPDATE PROCTYPE SET rptNature ='Spinal drug' WHERE PROCTYPE
UPDATE PROCTYPE SET rptNature ='IV access' WHERE PROCTYPE =
UPDATE PROCTYPE SET rptNature ='IV infusion' WHERE PROCTYPE
UPDATE PROCTYPE SET rptNature ='IV boluses' WHERE PROCTYPE =
UPDATE PROCTYPE SET rptNature ='IV PCA' WHERE PROCTYPE = 390
UPDATE PROCTYPE SET rptNature ='Transdermal drug' WHERE PROC
UPDATE PROCTYPE SET rptNature ='SC drug' WHERE PROCTYPE = 2
UPDATE PROCTYPE SET rptNature ='Per-rectal drug' WHERE PROCT
UPDATE PROCTYPE SET rptNature ='Nasal drug' WHERE PROCTYPE =
UPDATE PROCTYPE SET rptNature ='interscalene catheter' WHERE
UPDATE PROCTYPE SET rptNature ='interscalene infusion' WHERE
UPDATE PROCTYPE SET rptNature ='infraclavicular catheter' WH
UPDATE PROCTYPE SET rptNature ='infraclavicular infusion' WH
UPDATE PROCTYPE SET rptNature ='axillary catheter' WHERE PRO
UPDATE PROCTYPE SET rptNature ='axillary infusion' WHERE PRO
UPDATE PROCTYPE SET rptNature ='interpleural catheter' WHERE
UPDATE PROCTYPE SET rptNature ='interpleural infusion' WHERE
```

```
UPDATE PROCTYPE SET rptNature ='femoral catheter' WHERE PROC
UPDATE PROCTYPE SET rptNature ='femoral infusion' WHERE PROC
UPDATE PROCTYPE SET rptNature ='sciatic catheter' WHERE PROC
UPDATE PROCTYPE SET rptNature ='sciatic infusion' WHERE PROC
UPDATE PROCTYPE SET rptNature ='incisional catheter' WHERE P
UPDATE PROCTYPE SET rptNature ='incisional infusion' WHERE P
UPDATE PROCTYPE SET rptNature ='DATA' WHERE PROCTYPE = 1
UPDATE PROCTYPE SET rptNature ='ADMISSION' WHERE PROCTYPE =
UPDATE PROCTYPE SET rptNature ='24hr check' WHERE PROCTYPE =
```

All changes required are contained in the supplementary file *UPGRADE20080405.SQL*

3. There is a problem on the PC where we try to re-admit a patient. As things stand the response to item 86 (search for "WHERE iID = 86") means that a previous admission will be identified as current; this is not utterly beyond the pale, but we had to tweak the response to allow readmission! We did three things:

   (a) Set the **id** variable to the internal database ID of the patient, as required by ProcAndEpoch, called from within ReadmitPatient;

   (b) Modify the query to determine if there is a BADOBS that is both not inactive and not cold — if so, simply warn, but otherwise readmit;

   (c) M

```
UPDATE ITEM SET iResponse =
  'NULL->SET(hospitalnumber)->
   COPY->&GoodNhi->BOOLEAN->SKIP->=Fail(Invalid NHI)->
   UPPERCASE->COPY->SET(hospitalnumber)->
   QUERY(SELECT PERSDATA.pdoPerson FROM PERSDATA
   WHERE PERSDATA.pdoHospNo = ''$[]'')->
   QOK->SKIP->RETURN->
   COPY->SET(id)->
   QUERY(SELECT BADOBS.Bed
    FROM BADOBS WHERE BADOBS.Person = $[] AND BADOBS.boInactive
    AND BADOBS.cold IS NULL)->
   QOK->SKIP->=ReadmitPatient->
  #10000->DIV->INTEGER->
  QUERY(SELECT WARD.swrdText FROM WARD
    WHERE WARD.ward = $[])->
```

```
   Alert(Patient is in Ward $[]!)->
   POPMENU(#0)->DISCARD->DISCARD->MENU(0)'
  WHERE iID = 86;

UPDATE FUN SET fBody =
    'COPY->$[id]->DOSQL(UPDATE PROCESS SET cold=3 WHERE ProcType
    BURY->"Process"->KEY->copy->$[id]->now->now->me->DIGUP->
    DOSQL(INSERT INTO PROCESS
     (process,Person,rStart,rCreated,rPlanner,ProcType)
     VALUES($[],$[],TIMESTAMP ''$[]'',TIMESTAMP ''$[]'',$[],$[])
    "Epoch"->KEY->copy->BURY->SWOP->now->me->
    DOSQL(INSERT INTO EPOCH(epoch,Process,oMade,Person)
    VALUES($[],$[],TIMESTAMP ''$[]'',$[]))->DIGUP' WHERE fKey =

UPDATE FUN SET fBody = 'CONFIRM(Re-admit patient?)->SKIP->MENU(0
    #3->&ProcAndEpoch->BURY->
     "Badobs"->KEY->
     $[ward]->#10000->MUL->
    DIGUP->X->
      COPY->
      DOSQL(UPDATE BADOBS SET boInactive=1 WHERE Person = $[])->
    DOSQL(INSERT INTO BADOBS(badobs,Bed,Epoch,Person,boFlag)
    VALUES($[],$[],$[],$[],0))->
    #1->&ProcAndEpoch->
    MENU(DETAILS)'  WHERE fKey = 239;
```

The big question is ProcAndEpoch (function 112). If the process already exists and is not cold, then we should either inactivate that process, or simply return the old epoch! Now ProcAndEpoch is only used by CreateAdmission and ReadmitPatient. Thus, for this patient, we have ProcAndEpoch inactivate similar processes, if not already done.

WE HAVE A PROBLEM! ReadmitPatient at present requires that the person is in X (not the ward), despite ProcAndEpoch requiring the same value in $[id]. We must thus SetX to the patient ID, so we modify ReadmitPatient to do this.

Check that the upgrade functions on the PDA (NB sql). NO! Now none of admission, readmission or identification of current admission works. Item 86 is presumably stuffing things up. When we modify and export this item, even an Alert fails to occur??

The problem was a buffer length of 512 into which the new iResponse didn't fit [EXPLORE THIS 'SILENT' FAILURE — See CProgMain.tex relen and qlen (now 1024)].

4. Update 'New patients' button to go dark (toggle) if new patients present:

```
UPDATE ITEM SET iInitial = '$[activeW]->"new"->IN->NOT->SKIP->TO
```

5. Add IV paracetamol. The DRUGFORM entry is 4 (IV). The DRUG is new 'IV Paracetamol', a bit of a hack at 5120. We then reference this as 'IV bolus usage' with a drUsage value of 2. We manually generate the drugusage key.

```
INSERT INTO DRUG (drug, dTrade, DrugForm) VALUES (5120, 'IV para
INSERT INTO DRUGUSAGE (drugusage, Drug, drUsage) VALUES (56, 512
```

6. Exploration of modifications to FAIL verb. See also CProgMain.tex, notes for v 0.95. FAIL is used sparingly, by items 422 and 2422 to prevent their creation if inappropriate (both related to pethidine PCEA); by &GetAlert, which is only used by item 9923 with similar intent; but also in the &Fail routine which itself is invoked from many places. The 9923 item simply fails to create a checkbox in the Ward menu if the person has been discharged.

   &Fail is invoked as such only by SetTotal, as it is usually declared as =Fail (this is largely because of the current deficiency). If we examine all of these instances, (including SetTotal) the intention is for FAIL to completely fail. There thus seems to be no reason not to modify the functionality of FAIL in both Perl and PDA versions.

7. Altered KiwiDate to also accept a slash as the separator!

8. The entry (honest) menu now displays date and time, and provides an option to quit if the date/time are incorrect. This is mainly for use on the PDA, but also clearly has applicability on the desktop machine.