DataFax Edit Checks: Tips and Tricks

Darryl Pahl
Vice President
DF/Net Research, Inc.
Seattle, Washington
A Collection of Edit Check Examples

- DF/Net has created a variety of useful edit checks over the years – this is a sampling of some interesting techniques
- Some knowledge of edit check programming is helpful, but real objective is to provide ideas to take back home
A Collection of Edit Check Examples

- Overall flow is from general tips to easy tricks to more complicated edits that combine tips and tricks
- Enjoy!
Read the Documentation

- Each new version of DataFax enhances the edit check language
- *Read it!* Even those proficient in edit check programming find new ways to do things
- If you would like more/better documentation, please suggest improvements at:
  support@datafax.com
From Brian Postle, Database Manager at DF/Net:

“I used to be afraid of the Programmer Guide...Now I don’t hesitate to search for a key word, and I’m off and running. Although it seems intimidating at first, each topic/function is presented in short, bite-sized chunks of information. Once you dive in you’ll see that there’s a whole world to discover in there!”
Read the Documentation

- From Darryl Pahl, Vice President at DF/Net:
  “Yeah, it’s great”
**TIP**

**Generic Edit Checks**

- **General tip:** try to write edit checks as generically as possible.
  - This allows for:
    - Re-use for multiple studies as global edit checks
    - One-time validation, perhaps in a validation study
    - One place to fix things
edit Required() {
    string message1, message2;

    message1 = dfvarinfo(@T, DFVAR_DESC)
        + " is a required field, but currently it is blank. "
        + "Please complete, then initial, date, and re-fax the CRF."
    message2 = dfvarinfo(@T, DFVAR_DESC)
        + " is a required field, but currently it is blank."

    if (dfblank(@T)) {
        if (VERBOSE)
            dfaddqc(@T, 1, message1, 1, 2, "");
        else
            dfaddqc(@T, 1, message2, 1, 2, "");
    }
}

- Basic edit check to add to all required fields
  - Note that no variables are hard-coded; use @T and positional notations like "@[.+1], @T(-1)", etc.
  - Use of global variables like VERBOISE to control how QC looks (some studies like it verbose, some don’t)
  - Message text build using generic language
Interactive vs. Batch

- Are the edit checks designed to run interactively, in batch, or both?
  - Do any warnings, QCs, or messages change if run in batch?
  - Does the edit check even have meaning in batch?
  - Should it add QCs/change data in batch?
- New DataFax 2014.1 Batch Edits Tool replaces the need to create raw XML and control files with a Graphical User Interface
Using the 2014.1 Batch Edits Tool
Using the 2014.1 Batch Edits Tool

- Which records are of interest?
  - Select by meta data
  - Select by keys
  - Select by DRF
  - Select by edit check

- Result is intersection of criteria
Logging

- What edit check actions should be logged?
- Great for
  - Debugging (when=all)
  - Reviewing / confirming selection criteria
  - Keeping history / reducing future review work

![LOG interface](image.png)
Applying

- First, confirm behavior with LOG
- Then, should changes be applied to the database?
  - Which type(s) of changes?
- Update the validation level?

- There is no batch undo
Dynamic Legal Check

- **Difficulty Level:** *Simple*

- **Problem:**
  - Legal ranges appear everywhere; how to efficiently check and query them with a single generic check

- **Description:**
  - An edit check that uses the DFsetup definition description and legal range to dynamically create and apply a meaningful query
Assumptions:

- Do nothing if the variable is marked as missing
- Legal range is defined in setup as low~high, e.g., 3~7
- The variable description is consistent, standard, and meaningful
- Global variables exist that include:
  
  ```
  string s;
  number n;
  ```
- These will be used for typecasting
This movie has been modified from its original version. It has been formatted to fit your screen.
edit LegalRange() {
    string message, legal_range = dfvarinfo(@T, DFVAR_LEGAL);

    # Build QC message for legal range
    message = dfvarinfo(@T, DFVAR_DESC)
    + " has a value of "
    + (s=@T)
    + ", which is inconsistent with our expected range of "
    + legal_range
    + ". Please verify that this value is correct.";

    # See if the variable is in the range
    if (!dflegal(@T)) {
        # If it is, and this is being run in batch, and the
        # value is resolved, print a warning message for the
        # batch log along with any query note
        if (dfbatch() && dfresqc(@T))
            dfwarning(message + " RESOLVED: " +
                       dfqcinfo(@T,DFQCNOTE));
        # Add a query with the dynamic query message text
        dfaddqc(@T, 3, message, 1, 2, "");
    }
}
edit LegalRange() {
    string message, legal_range = dfvarinfo(@T, DFVAR_LEGAL);

    # Build QC message for legal range
    message = dfvarinfo(@T, DFVAR_DESC) + " has a value of " + (s=@T) + ", which is inconsistent with our expected range of " + legal_range + ".
    # See if the variable is in the range
    if (!dflegal(@T)) {
        # If it is, and this is being run in batch, and the value is resolved, print a warning message for the batch log along with any query note
        if (dfbatch() && dfresqc(@T))
            dfwarning(message+ " RESOLVED: " + dfqcinfo(@T,DFQCNOTE));
        # Add a query with the dynamic query message text
        dfaddqc(@T, 3, message, 1, 2, "");
    }
}
edit LegalRange() {
    string message, legal_range = dfvarinfo(@T, DFVAR_LEGAL);

    - Standard header and declarations
      - This is an edit check called LegalRange
      - Two strings are declared, message and legal range
      - Legal range must be defined in the setup like this:

    Legal: low~high
    or range
    or list
Keep Edit Checks Simple

- There are many combinations of specifications that we receive
  - “if illegal”
  - “must not be missing”
  - “must not be blank”
- Write generic edit checks with a singular purpose, and then string them together in setup
One edit check could determine if age was legal, and either age or DOB were required, and age was not required if DOB was there... but it would not be very generic

Instead, create three that are generic and do the same checks serially
edit LegalRange() {
    string message, legal_range = dfvarinfo(@T, DFVAR_LEGAL);

    # Build QC message for legal range
    message = dfvarinfo(@T, DFVAR_DESC)
    + " has a value of "
    + (s:@T)
    + ", which is inconsistent with our expected range of "
    + legal_range
    + ". Please verify that this value is correct."

    # See if the variable is in the range
    if (!dflegal(@T)) {
        # If it is, and this is being run in batch, and the
        # value is resolved, print a warning message for the
        # batch log along with any query note
        if (dfbatch() && dfresqc(@T))
            dfwarning(message+ " RESOLVED: " +
                      dfqcinfo(@T,DFQCNOTE));
        # Add a query with the dynamic query message text
        dfaddqc(@T, 3, message, 1, 2, "")
    }
}
# Build QC message for legal range
message = dfvarinfo(@T, DFVAR_DESC)
+ " has a value of ",
+ (s=@T)
+ ", which is inconsistent with our expected range of ",
+ legal_range
+ ". Please verify that this value is correct.";

- **Note the use of casting:**
  - Adding string fields together results in them being concatenated (strung together)
  - But you can only add strings, not strings and numbers
  - “s” is a global variable that we define and use across all edit checks to save time
  - Also define:
    ```
    string s;
    number n;
    date d;
    ```
Type Casting in DataFax

- Note the use of casting:
  - Type casting allows variables to be molded into a different variable type
What You Get Without Type Casting

In `/studynas/IPM_027_2014/ecsrc/DFedits`, line 17 (+): incompatible types for '+' operator
# Build QC message for legal range
message = dfvarinfo(@T, DFVAR_DESC)
+ " has a value of 
+ (s=@T)
+ ", which is inconsistent with our expected range of 
+ legal_range
+ ". Please verify that this value is correct."

- Build a query message using the value in the description field in setup:
  Description

- Also see how strings can be added
edit LegalRange() {
    string message, legal_range = dfvarinfo(@T, DFVAR_LEGAL);

    # Build QC message for legal range
    message = dfvarinfo(@T, DFVAR_DESC)
        + " has a value of "
        + (s=@T)
        + ", which is inconsistent with our expected range of "
        + legal_range
        + ". Please verify that this value is correct."
;

    # See if the variable is in the range
    if (!dflegal(@T)) {
        # If it is, and this is being run in batch, and the
        # value is resolved, print a warning message for the
        # batch log along with any query note
        if (dfbatch() && dfresqc(@T))
            dfwarning(message + " RESOLVED: " +
                dfqcinfo(@T, DFQCNOTE));
        # Add a query with the dynamic query message text
        dfaddqc(@T, 3, message, 1, 2, "");
    }
}
# See if the variable is in the range
if (!dflegal(@T)) {
    # If it is, and this is being run in batch, and the value
    # is resolved, print a warning message for the batch log
    # along with any query note
    if (dfbatch() && dfresqc(@T))
        dfwarning(message + " RESOLVED: " + dfqcinfo(@T, DFQCNOTE));
    # Add a query with the dynamic query message text
    dfaddqc(@T, 3, message, 1, 2, "");
}

- **Time to do the real work**
  - The “if” statement sees if the value is out of range

- **Are we in batch? And is there a resolved QC?**
  - If yes, then print a warning message
  - If we are in interactive, the warning will never be received
  - But the warning, and query note, will appear in the batch report; useful for people to know
Useful Batch Concepts

- Add useful dfwarning() messages for use in batch
  - In the example, variables may be out of range, but have resolved QC with a note in the explanation
  - When run in batch, a reviewer can see that the value is illegal/inconsistent, and see that there is a note attached to it
# See if the variable is in the range
if (!dflegal(@T)) {
    # If it is, and this is being run in batch, and the value
    # is resolved, print a warning message for the batch log
    # along with any query note
    if (dfbatch() && dfresqc(@T))
        dfwarning(message + " RESOLVED: " + dfqcinfo(@T, DFQCNOTE));
    # Add a query with the dynamic query message text
    dfaddqc(@T, 3, message, 1, 2, "");
}

- dfaddqc() will add a query with the constructed message (or do nothing if there is one there)

- The edit check can be applied on field exit, for example:
Putting It All Together: Interactive
Putting It All Together: Interactive

Demographics

1. Date of birth:  
   - OR Age: 46 years

2. Race:

3. Education:
   - 3a. Primary school completes
   - 3b. Secondary school completes
   - 3c. Tertiary education completed

Site: 1, Patient: 100001, Assessment: Screen Failure, Page: DEM, Screen Failure

Field: Age (item 1)

Reported Value: 46

Problem: Inconsistent

Status: New query

Details: Age (item 1) has a value of 46, which is inconsistent with our expected range of 18~45, 60~99. Please verify that this value is correct.
Be Mindful of QC Wording

- Perhaps programmers are not the best to design QC language...
  - Don’t use leading language. Use “Please verify that this is correct” vs. “Please correct”
  - If appropriate, use “inconsistent” vs. “illegal”
  - Maybe the value is correct, but your range is wrong
Be Mindful of QC Wording
Making Edit Checks Faster

- Consider ways to make edit checks run faster by reducing or rearranging steps
- For example:
```c
edit LegalRange() {
    string message, legal_range = dfvarinfo(@T, DFVAR_LEGAL);

    # Build QC message for legal range
    message = dfvarinfo(@T, DFVAR_DESC)
        + " has a value of "
        + (s=@T)
        + ", which is inconsistent with our expected range of "
        + legal_range
        + ". Please verify that this value is correct."
;

    # See if the variable is in the range
    if (!dflegal(@T)) {
        # If it is, and this is being run in batch, and the value is resolved, print
        # a warning message for the batch log along with any query note
        if (dfbatch() && dfresqc(@T))
            dfwarning(message+ " RESOLVED: " + dfqcinfo(@T,DFQCNOTE));
        # Add a query with the dynamic query message text
        dfaddqc(@T, 3, message, 1, 2, "");
    }
}
```

- `dfvarinfo` and building the message get called even if the value is fine
- Try to reorganize to save steps if the logic is not called
```c
edit LegalRange() {
  string message, legal_range;

  # See if the variable is in the range
  if (!dflegal(@T)) {
    # If it is, and this is being run in batch, and the value is resolved, print
    # a warning message for the batch log along with any query note

    # Build QC message for legal range
    message = dfvarinfo(@T, DFVAR_DESC)
    + " has a value of "
    + (s=@T)
    + ", which is inconsistent with our expected range of "
    + dfvarinfo(@T, DFVAR_LEGAL)
    + ". Please verify that this value is correct."
;

    if (dfbatch() && dfresqc(@T))
      dfwarning(message+ " RESOLVED: " + dfqcinfo(@T,DFQCNOTE));
    # Add a query with the dynamic query message text
    dfaddqc(@T, 3, message, 1, 2, "");
  }
```

- Message is only created if value is illegal
- Don’t even need the variable “legal_range”
- Sometimes there is a compromise between efficiency and readability
Extra Credit

- The dfwarning could better differentiate between resolved and unresolved queries
- Currently, no query is added if one is there, logic could be added to better address this
DFopen_study() Use

- Difficulty Level: *Simple*
- A collection of ideas for what you might set/change/lock when a study is first opened
- Not implying that all are good ideas, just examples that could be adapted for your use!
Setting the iDataFax View

- When the study is first opened, you can ask questions and change DataFax user preferences:

![Image of the iDataFax - dfask: DFopen_study dialog box with options for dashboard or data view.]
edit DFopen_study()
{
    number screen;

    screen = dfask("<center>Would you like to see the dashboard or data view?"","1","Dashboard","Data View");
    if (screen == 1)
        dfpref("OpenDashBoard", "Yes", DFPREF_SESSION);
    else
        dfpref("OpenDashBoard", "No", DFPREF_SESSION);
}
Setting the iDataFax View
Setting Various Preferences

- Can set or lock various preferences when the study is opened
- Used to create a consistent user interface without the need to teach users how to set preferences
  - For example, if your role is a site monitor, always show the dashboard
  - If you are a DF/Net person, just go to data view
User Preferences

- Most user preferences can be set or locked when the study is opened
- For example:
Use of dfpref()

- dfpref() takes 3 parameters:
  - prefname - one of the iDataFax user preferences
  - prefvalue - one of the settable values for the preference
  - duration - how long the preference setting lasts, which may be:
    - DFPREF_CURRENT = current page only
    - DFPREF_SESSION = current user session
    - DFPREF_LOCK = always, can not be modified by the user
edit DFopen_study()
{
    dfpref("ShowVisit","Label",DFPREF_SESSION);
    dfpref("ShowPlate","Label",DFPREF_SESSION);
    dfpref("ShowSite","NumberLabel",DFPREF_SESSION);
    dfpref("AutoLogout","18",DFPREF_LOCK);
}

- Show visits as labels
- Show plates as labels
- Show sites as number – labels
- Use the patient binder format
- Lock the Autologout to 18 minutes
User Preferences

- Note that auto logout is locked

```
dfpref("ShowVisit", "Label", DFPREF_SESSION);
dfpref("ShowPlate", "Label", DFPREF_SESSION);
dfpref("ShowSite", "NumberLabel", DFPREF_SESSION);
```

```
dfpref("AutoLogout", "18", DFPREF_LOCK);
```
Fun with Pop-Ups

- Difficulty Level: *Simple or more difficult*
- A collection of ideas for use of dfwarning(), dfask(), and dfcapture()
Use of Warning()

- `dfwarning(expn1, expn2, ...)`
  - `dfwarning` takes one or more expressions of any data type
- A simple example showing the use of `dfwarning` and DEBUG
- DEBUG is a global variable that we sometimes use before edit checks
- When there’s trouble, you can turn on debugging and see what is going on
# Global variables
string DEBUG = "darryl";

edit MyExample() {
    date start_date, end_date;
    # This is the end date
    end_date = @T;
    # The start date is in the previous field
    start_date = @(T-1);

    if (dfwhoami() == DEBUG)
        dfwarning("Start Date is: ", start_date,
            " End Date is: ", end_date,
            " Days Difference is: ", start_date-end_date);

    # Do a bunch of other important stuff here
    # ...
}
Create a Global DEBUG Variable

- By setting a global variable:

  ```
  string DEBUG = "darryl";
  ```

- You can write edit checks with various debug logic to display information

  ```
  if (dfwhoami() == DEBUG)
  ```

- Only trigger if “darryl” is running the edit

  ```
  # only user darryl
  string DEBUG = "darryl";
  # turn off debugging for everyone
  string DEBUG = "";
  ```
# Global variables
string DEBUG = "darryl";

edit MyExample() {
    date start_date, end_date;
    # This is the end date
    end_date = @T;
    # The start date is in the previous field
    start_date = @(T-1);
    
    if (dfwhoami() == DEBUG)
        dfwarning("Start Date is: ", start_date,
                   " End Date is: ", end_date,
                   " Days Difference is: ", start_date-end_date);

    # Do a bunch of other important stuff here
    # ...
}

A Boolean test is less expensive than a string compare so perhaps a study open edit check that does the dfwhoami() test and then sets the Boolean DEBUG variable. Or, for certain roles, perhaps ask during study open if the user would prefer debug mode?
# Global variables
string DEBUG = "darryl";

edit MyExample() {
    date start_date, end_date;
    # This is the end date
    end_date = @T;
    # The start date is in the previous field
    start_date = @(T-1);

    if (dfwhoami() == DEBUG)
        dfwarning("Start Date is: ", start_date,
                  " End Date is: ", end_date,
                  " Days Difference is: ", start_date-end_date);

    # Do a bunch of other important stuff here
    # ...
}

Make Code Readable

- Use newlines to make the code more readable

*Compare:*

```python
dfwarning("Start Date is: ", start_date, ", End Date is: ", end_date, ", Days Difference is: ", start_date-end_date);
```

*To:*

```python
dfwarning("Start Date is: ", start_date, 
" End Date is: ", end_date, 
" Days Difference is: ", start_date-end_date);
```
Make Messages More Fun

- You can use HTML code to enhance the user experience:

  Functional, but dull:
Make Messages More Fun

- You can use HTML code to enhance the user experience: Red for Bad Date
Make Messages More Fun

- You can use HTML code to enhance the user experience: Green for Good Date

This is a basic HTML code snippet to show the “Green for Good Date”
if (start_date - end_date <= 0) {
    dfwarning("<HTML><HEAD><STYLE>
    .g {background-color: #799; color: #fff; padding: 2px; text-align: right;}
    .h {background-color: #fff; padding: 2px; text-align: right;}
    .i {background-color: #A8ECB2; padding: 2px; text-align: right;}
    </STYLE></HEAD>
    <BODY><DIV>
    <P><big>Debugging Messages</big></P>
    <TABLE BORDER=0 MARGIN=0>
    <TR><TD CLASS=g>Start Date:</TD><TD CLASS=h>
    start_date,
    </TD></TR>
    <TR><TD CLASS=g>End Date:</TD><TD CLASS=h>
    end_date,
    </TD></TR>
    <TR><TD CLASS=g>Days Difference:</TD><TD CLASS=i>
    start_date-end_date,
    </TD></TR>
    </TABLE>
    </P><big>Good Date!</big></P>
    <P>The start date is before the end date</P>
    </DIV></BODY></HTML>\n"};
}
Use Global Variables as Containers

- If you have repeating text that may get used in many places, create a global variable that contains it
- Then, you can call the variable rather than repeating the text
- Only need to change it in one place!
string HTML_HEADING = "HTML><HEAD><STYLE>
 + ".g {background-color: #799; color: #fff; padding: 2px; text-align: right;}
 + ".h {background-color: #fff; padding: 2px; text-align: right;}
 + ".i {background-color: #A8ECB2; padding: 2px; text-align: right;}
 + "</STYLE></HEAD>\n";

HTML Color Codes

- An excellent resource for HTML color codes is:

http://www.computerhope.com/htmcolor.htm
Designing HTML

- I’m no HTML expert
- Just copy something that works, and start messing with it
- I used the example in the Edit Check documentation

*(see Tip #1: Read the Documentation)*
Dynamic Visit Labels

- **Difficulty Level:** Simple
- **Problem:**
  - In the patient binder, it would be nice if pages could have meaningful, dynamic names based on data
- **Description:**
  - Use of the DataFax page map can make this happen
Dynamic Visit Labels

- Not really an edit check, but still cool

- In the DataFax page map, you can map plate and sequence to human readable labels
Dynamic Visit Labels

- Note the use of: 
  
  \(\text{AE,}\%\{S.2\}\%10\)

  This means the label should start with “AE,”
Dynamic Visit Labels

- Note the use of: AE, \( \%\{S.2\} \) \%10

This means use that \( \%\{S.n\} \) is replaced by the last \( n \) digits of the zero padded sequence number.
Dynamic Visit Labels

- Note the use of: `AE, %{S.2} %10`

This means use that `%10` is replaced by the value contained in field 10 on the form.
Putting It Together

- This makes the AE (field 10) appear as the page label
- You can use any field, including a field that you generate on the fly
Creating a Label From Multiple Fields

 Dummy field created by an edit check

 Results in:
Using `dfexecute()`

- Used to call programs external to DataFax
- Must be executable and live in the `$(STUDY_DIR)/ecbin` directory
- Called with:
  ```
  dfexecute("Program", Parameter List)
  ```

For example:
Using dfexecute()

# both of the following examples pass 3 parameters
# to myscript.sh

string s;

s = dfexecute( "myscript.sh", "10156 2 1" );
s = dfexecute( "myscript.sh", "10156", "2", "1" );
dfexecute() Gotchas

- Passed arguments can’t include:
  - Characters which have special meaning to the UNIX shell will not be allowed in the command line submitted, including $, ‘, ;, |, <, and >.

- If the ampersand (&) character is passed anywhere in the parameter list, the parameter list will be truncated where the ampersand was encountered
dfexecute() Gotchas

- If you have “bad” characters in your parameter list, you may see this:
dfexecute() Gotchas

- If you have not put the program in ecbin, or are calling it incorrectly, you may see this:
dfexecute() Gotchas

- And sometimes you don’t see anything at all ..... (!)
Using dfmessage() to Debug dfexecute()

- You can use dfmessage() to track the output, including any error messages of dfexecute()
- For example, if running:
  
  ```sh
  dfexecute("pillsy.sh");
  ```

  Results in:
Using dfmessage() to Debug dfexecute()

- Try changing that statement to:

  ```csharp
dfmessage(dfexecute("pillsy.sh"));
  ```

Results in a more useful:
Using `dfmessage()` to Debug `dfexecute()`

- I need to fully qualify the Python script “parse_json.py”
Another Example of dfexecute()

- Some things are hard to do in DataFax

- External programs may make this easier, for example, returning the current ISO 8601 date in UTC (Coordinated Universal Time)
Another Example of dfexecute()

- For example, now is:
  2015-09-26T23:01:28Z
  year 2015, month 09, day 26, time is 11:01:28 PM at 0° longitude

- Difficult to determine in DataFax, easy in Perl:
Return ISO Time/Date in UTC

#!/bin/perl

use POSIX qw(strftime);

# ISO formatted date in UTC
$datestring = strftime "%FT%Z", gmtime;
printf($datestring);
Return ISO Time/Date in UTC

Current_ISO.UTC.pl as located in ecbin:

#!/bin/perl

use POSIX qw(strftime);

# ISO formatted date in UTC
$datestring = strftime "%FT%XZ", gmtime;
printf($datestring);
# Return the current ISO UTC Date and Time

string iso_date_time;

iso_date_time = dfexecute("Current_ISO.UTC.pl");

iso_date_time ends up with:

2015-09-26T23:01:28Z
Using dfexecute(), DF_open_patient_binder(), and dftrigger() to Build Records on the Fly

- **Difficulty Level:** *Just by the title, a little less simple*

- **Problem:**
  - Dynamically add records to DataFax based on external data sources whenever a patient binder is opened

- **This is just a proof of concept!**
  - May not be practical since the trigger is interactive
  - Alternative to loading records via DFimport.rpc and cron in batch
What We Want

- When a binder is opened, the records should be created on the fly, like:
What We Want

- Each event should have a new visit created
### What We Want

- Each plate should be shown with the event type, date, and time as a label.
```
string s;
date d;
date format "mm/dd/yy"
string DEBUG="darryl";

edit DFopen_patient_binder() {   # edit check gets run every time the patient binder is opened
    string event_list, event;
    number visit, plate, ptid, line=1;
    dfexecute("pillsy.sh");
    event_list = dfexecute("GetRecords.sh", @PID);
    event = dfgetfield(event_list, line, "\n");
    while(event) {
        visit = dfgetfield(event, 1, ",");
        ptid = dfgetfield(event, 2, ",");
        if (ptid == @PID) {
            dftrigger(@PID, visit, 1, "1", "1", "LoadData", 0);
        }
        line = line + 1;
        event = dfgetfield(event_list, line, "\n");
    }
}

edit LoadData() {   # edit check that gets called on plate enter to fill in the data
    string match, event_data;
    number visit, plate, ptid;
    dfautoreason(0);
    match = "" + (s=@[6]) + "," + (s=@[7]) + ""
    event_data = dfexecute("GetData.sh", match, (s=@[7]));
    @[8] = dfgetfield(event_data, 3, ",");
    @[9] = (d=dfgetfield(event_data, 4, ","));
    @[10] = dfgetfield(event_data, 5, ",");
    @[11] = dfgetfield(event_data, 6, ",");
    @[12] = dfgetfield(event_data, 7, ",");
    @[13] = dfgetfield(event_data, 8, ",");
    @[14] = @[8] + ": " + (s=@[9]) + " " + (s=@[10]);
```
**DFopen_patient_binder**

- Built-in edit check that, if it exists, is executed every time the patient binder is opened.
- Often used to set preferences, to show/hide plates and visits.
- Also plays well with dftrigger(), dfneed(), DFopen_study, dfrole(), dflogout().
- We are going to use this as our magic trigger event to get data.
DFopen_patient_binder()

```c
edit DFopen_patient_binder() {
    string event_list, event;
    number visit, plate, ptid, line=1;
    dfexecute("pillsy.sh");
    event_list = dfexecute("GetRecords.sh", @PID);
    event = dfgetfield(event_list, line, "\n");
    while(event) {
        visit = dfgetfield(event, 1, ",");
        ptid = dfgetfield(event, 2, ",");
        if (ptid == @PID) {
            dftrigger(@PID, visit, 1, "1", "1", "LoadData", 0);
        }
        line = line + 1;
        event = dfgetfield(event_list, line, "\n");
    }
}
```
DFopen_patient_binder()

```c
edit DFopen_patient_binder() {
    string event_list, event;
    number visit, plate, ptid, line=1;
    dfexecute("pillsy.sh");
    event_list = dfexecute("GetRecords.sh", @PID);
    event = dfgetfield(event_list, line, ",\n");
    while(event) {
        visit = dfgetfield(event, 1, ",");
        ptid = dfgetfield(event, 2, ",");
        if (ptid == @PID) {
            dftrigger(@PID, visit, 1, "1", "1", "LoadData", 0);
        }
        line = line + 1;
        event = dfgetfield(event_list, line, ",\n");
    }
}
```

Will run every time the patient binder is opened.
DFopen_patient_binder()

This is the script written by Saloni to talk to the Pillsy cloud to get all events.

Populates files in the work directory, one file per pill cap (e.g.: $STUDY_DIR/work/9990001.csv, 9990002.csv, etc.)

```c
edit DFopen_patient_binder() {
    string event_list, event;
    number visit, plate, ptid, line=1;
    dfexecute("pillsy.sh");
    event_list = dfexecute("GetRecords.sh", @PID);
    event = dfgetfield(event_list, line, "\n");
    while(event) {
        visit = dfgetfield(event, 1, ",");
        ptid = dfgetfield(event, 2, ",");
        if (ptid == @PID) {
            dftrigger(@PID, visit, 1, "1", "1", "LoadData", 0);
        }
        line = line + 1;
        event = dfgetfield(event_list, line, "\n");
    }
}
```
This returns a newline delimited list of events for this patient, using the special keyword @PID
DFopen_patient_binder() {
    string event_list, event;
    number visit, plate, ptid, line=1;
    dfexecute("pillsy.sh");
    event_list = dfexecute("GetRecords.sh", @PID);
    event = dfgetfield(event_list, line, \\n);
    while(event) {
        visit = dfgetfield(event, 1, ",");   
        ptid = dfgetfield(event, 2, ",");
        if (ptid == @PID) {
            dftrigger(@PID, visit, 1, "1", "1", "LoadData", 0);
        }
        line = line + 1;
        event = dfgetfield(event_list, line, \\n);
    }
}

Start splitting these events up into individual events, all events are on a line by themselves, so use dfgetfield with a "\n" newline delimiter
dfgetfield(event_list, line, "\n");

- That’s worth a pause
- The file 999001.csv looks like:

```
101,999001,Connected to Pillsy.,09/25/15,15:01:21,Crestor,47.5951865,-122.2700496
102,999001,Pillsy close,09/25/15,15:01:59,Crestor,47.5956004,-122.2709413
103,999001,Pillsy open,09/25/15,15:01:59,Crestor,47.5956004,-122.2709413
104,999001,Pillsy close,09/25/15,15:02:07,Crestor,47.5956004,-122.2709413
105,999001,Pillsy open,09/25/15,15:02:07,Crestor,47.5956004,-122.2709413
106,999001,Pillsy close,09/25/15,15:02:09,Crestor,47.5956004,-122.2709413
107,999001,Pillsy open,09/25/15,15:02:10,Crestor,47.5956004,-122.2709413
```
Many cases where you want to read data from a file, and there is no file I/O in DataFax

But even a simple shell script like:

```bash
#!/bin/sh
cat $1
```

can be used to read in a file from the work directory, which is then parsed with something like:
filename = "999001.csv";
event_list = dfexecute("ReadFile.sh", filename);

event = dfgetfield(event_list, line, "\n");
while(event) {
    (do a bunch of stuff)
    line = line + 1;
    event = dfgetfield(event_list, line, "\n");
}

- If the file is empty, then the first event is null, and the while loop never runs
DFopen_patient_binder()

edit DFopen_patient_binder() {
    string event_list, event;
    number visit, plate, ptid, line=1;
    dfexecute("pillsy.sh");
    event_list = dfexecute("GetRecords.sh", @PID);
    event = dfgetfield(event_list, line, \
"n");
    while(event) {
        visit = dfgetfield(event, 1, ",");
        ptid = dfgetfield(event, 2, ",");
        if (ptid == @PID) {
            dftrigger(@PID, visit, 1, "1", "1", "LoadData", 0);
        }
        line = line + 1;
        event = dfgetfield(event_list, line, \\n"n");
    }
}

@PID is special DataFax value that is the current patient binder ptid

If the ptid matches this patient, then use dftrigger() to create a new record and add it immediately to the visible list
dftrigger() can be used to add a new record to the current patient binder and/or to execute all or specified plate entry edit checks on a specified data record in the current patient binder.

This provides a mechanism for creating conditional plates and updating data fields on other records that depend on values entered and saved on the current record.
dftrigger(@PID, visit, 1, "1", "1", "LoadData", 0);

- @PID is the current ptid
- visit is the Pillsy sequence number
- 1 is the plate # that we are creating
- "1" is for "final"
- "1" is for a level 1 record
- "LoadData" is the name of the plate enter edit check that we want to run after this record is created
- 0 is on of 0,1,2,3,4,5 actions (see the documentation)
- In our case 0 = Run the edit checks but do not open the data record
DFopen_patient_binder()

```
edit DFopen_patient_binder() {
    string event_list, event;
    number visit, plate, ptid, line=1;
    dfexecute("pillsy.sh");
    event_list = dfexecute("GetRecords.sh", @PID);
    event = dfgetField(event_list, line, "\n");
    while(event) {
        visit = dfgetField(event, 1, ",");
        ptid = dfgetField(event, 2, ",");
        if (ptid == @PID) {
            dftrigger(@PID, visit, 1, "1", "1", "LoadData", 0);
        }
        line = line + 1;
        event = dfgetField(event_list, line, "\n");
    }
}
```

Keep doing this for all records, i.e., loop through the data file
From DFopen_patient_binder:

```bash
loadtrigger(@PID, visit, 1, "1", "1", "LoadData", 0);
```

```bash
edit LoadData() { # Edit check assigned on plate enter on the ptid
  string match, event_data;
  number visit, plate, ptid;
  dfautoreason(0);
  match = "'" + (s=@[6]) + "," + (s=@[7]) + "'";
  event_data = dfexecute("GetData.sh", match, (s=@[7]));
  @[8] = dfgetfield(event_data, 3, ",");
  @[9] = (d=dfgetfield(event_data, 4, ","));
  @[10] = dfgetfield(event_data, 5, ",");
  @[11] = dfgetfield(event_data, 6, ",");
  @[12] = dfgetfield(event_data, 7, ",");
  @[13] = dfgetfield(event_data, 8, ",");
  @[14] = @[8] + ": " + (s=@[9]) + " " + (s=@[10]);
}
```

Now that we have a new, blank record from DFopen_patient_binder, let's populate it with data from the file. Use GetData.sh to return the actual data record that fits in here.
LoadData()
edit LoadData() {
    string match, event_data;
    number visit, plate, ptid;
    dfautoreason(0);
    match = "" + (s=@[6]) + "," + (s=@[7]) + "";
    event_data = dfexecute("GetData.sh", match, (s=@[7]));
    @[8] = dfgetfield(event_data, 3, ",");
    @[9] = (d=dfgetfield(event_data, 4, ","));
    @[10] = dfgetfield(event_data, 5, ",");
    @[11] = dfgetfield(event_data, 6, ",");
    @[12] = dfgetfield(event_data, 7, ",");
    @[13] = dfgetfield(event_data, 8, ",");
    @[14] = @[8] + ": " + (s=@[9]) + " " + (s=@[10]);
}

Just get the fields and fill them in where they should go in the data record
edit LoadData() { # Edit check assigned on plate enter on the ptid
    string match, event_data;
    number visit, plate, ptid;
    dfautoreason(0);
    match = "" + (s=@[6]) + "," + (s=@[7]) + ";";
    event_data = dfexecute("GetData.sh", match, (s=@[7]));
    @[8] = dfgetfield(event_data, 3, ",");
    @[9] = (d=dfgetfield(event_data, 4, ","));
    @[10] = dfgetfield(event_data, 5, ",");
    @[11] = dfgetfield(event_data, 6, ",");
    @[12] = dfgetfield(event_data, 7, ",");
    @[13] = dfgetfield(event_data, 8, ",");
    @[14] = @[8] + ": " + (s=@[9]) + " " + (s=@[10]);
}

This field is a date, so be sure to set your date format with this at the top of DFedits:
date format "mm/dd/yy"
edit LoadData() {# Edit check assigned on plate enter on the ptid
    string match, event_data;
    number visit, plate, ptid;
    dfautoreason(0);
    match = "'" + (s=@[6]) + "," + (s=@[7]) + "'";
    event_data = dfexecute("GetData.sh", match, (s=@[7]));
    @[8] = dfgetfield(event_data, 3, ",");
    @[9] = (d=dfgetfield(event_data, 4, ","));
    @[10] = dfgetfield(event_data, 5, ",");
    @[11] = dfgetfield(event_data, 6, ",");
    @[12] = dfgetfield(event_data, 7, ",");
    @[13] = dfgetfield(event_data, 8, ",");
    @[14] = @[8] + ": " + (s= @[9]) + " " + (s= @[10]);
}

This is a “dummy” field that looks like event: date time. We will use this to make the page label look nice.
Field 14 will be mapped into what is seen in iDataFax
Putting It All Together

- Every time the patient is opened, the Pillsy cloud is consulted, data downloaded, and records added dynamically while you watch.
Other Examples

- Instead of raw records, data could be extracted and a current adherence projection calculated.

- Other APIs could talk to labs, devices, or smart phone apps for patient diaries, reactogenicity smart apps, AE tracking, or SAE reconciliations.
Questions?

- A lot to cover!