Pournami Ver 0.3.3
Forms Ver 0.2.4
Guide

What's new in 0.2.4

In 0.2.4 two new additional data items to the list LARGETEXT, MONTHYEAR. The PForms has been made into a library to implement the UI in your own way, once the form's elements are read.

Introduction

Pournami, as you know by now, not only supports text editing and compiling by external tools, but also forms in itself. Now to answer all your question about Forms, please read through the next sections

What is Form

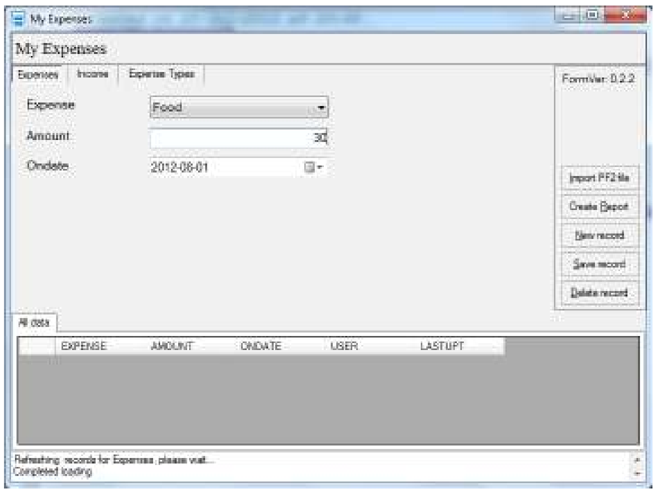
Form is nothing but a user interface to make data entry quickly and in required fashion. For example a login form, a registration form etc.

What are Pournami Forms

To ease the creation of forms, Pournami Forms have been created. The built in pseudo language is called "Pournami Forms Defintion Language" or PDFL. You can create a complex form in as short as 5 minutes. Yes it is true. Look at the following lines.

```
[META:My Expenses]

[FORMDEF:Expenses:EXPENSES]

[FIELD:EXPENSE:ROCOMBO:SELECT EXPNAME FROM EXPENSETYPES ORDER BY
EXPNAME]

[FIELD:AMOUNT:NUMERIC]

[FIELD:ONDATE:DATE]

[FORMDEF:Income:Income]

[FIELD:MONTH:DATE]

[FIELD:TYPE:LIST:SALARY,RENT,PROFITS,RETURNS,OTHER]

[FIELD:AMOUNT:NUMERIC]

[FORMDEF:Expense Types:ExpenseTypes]

[FIELD:EXPNAME:TEXT]
[FIELD:EXPFREQ:LIST:DAILY,WEEKLY,FORTNIGHTLY,MONTHLY,QUARTERLY,HALF
YEARLY,ANNUAL]
```

The above few lines would create a form like this.



My Expenses

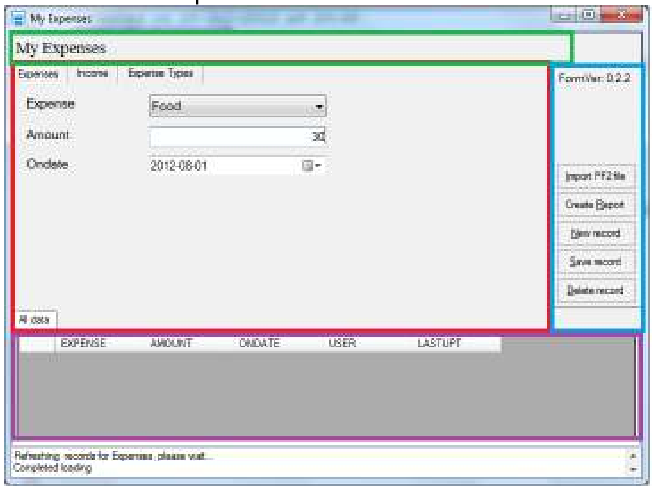
FormVer: 0.2.2

Expense: Food
Amount: 30
Ondate: 2012-08-01

EXPENSE AMOUNT ONDATE USER LASTUPT

Refreshing records for Expenses, please wait...
Completed loading

Now let me explain the elements of the form, taking the above example



My Expenses

FormVer: 0.2.2

Expense: Food
Amount: 30
Ondate: 2012-08-01

EXPENSE AMOUNT ONDATE USER LASTUPT

Refreshing records for Expenses, please wait...
Completed loading

Look at the enclosures above
Green: Title of the Form collection
Blue : Control area (Add, modify, delete records)
Purple: Records grid and reports
Red : Your forms collection.

You need to understand that the Interface look and feel will remain the same however big or complex your form is. Only the forms you defined will change inside the "RED" enclosure.

Let us see how we can set the form in the above interface.
Every form can be created using tags enclosed in square brackets "[" and "]" and delimited by colon ":" for its attributes.

There are very few tags to remember… Look at the following

1) [META:<Application Name>:<optional Username>]

In the above example see [META: My Expenses]. The user name is omitted here. Remember that a form's data has to be owned by a user, whether or not a form is owned by him or not. So a user name is mandatory while entering data. During compilation, whenever the form is executed for the first time, it asks username. So if you want to avoid such problem during form development, you can add your Username to META tag itself. But do remove it when you finally compile the form and send it to your team. Spaces are permitted in the name of the Application.

2) [FORMDEF:<Form name>:<Datastore>]

The form definition is by using tag FORMDEF, which identifies the form name and the datastore name in which data should be stored. There are no other variants of FORMDEF at this point of time. Each form should have one FORMDEF tag, under which the fields should be specified. The datastore is the place where all the data when entered, should be stored. You need not define anything else as a part of datastore. IT can be a simple name like "PROJECTS" or "TRANSACTIONS. Spaces are permitted in the form name, but not in the datastore name.

3) [FIELD:<field name>:<Field type>:<optional data source>]

Fields should fall under the FORMDEF and cannot be orphans. Each field can take a field name(no spaces). A field type of the following types.

a) **TEXT** : A plain text of maximum length 200 chars
b) **MULTILINE**: A multiline text of 32K characters
c) **LIST**:A list of specified comma delimited values
d) **LARGETEXT:** Same as multiline, but a bigger UI element to comfortably type.
e) **RICHTEXT:** There would be times when you want to use rich text instead of plain text. So this option allows you to have rich text editing capabilities in your form.
f) **ROCOMBO**: A fixed list of values from a datastore defined inside the application
g) **COMBO**: A list of values from the datastore defined inside the application, but also a facility to add new values
h) **CHECK**: A check box for flags or multiple options
i) **RADIO** : A radio button, for single options
j) **FILE** : A file to be attached
k) **PICTURE** : A picture to be attached
l) **NUMERIC** : A Numerical text box with right alignment
m) **DATE** : A date picker for showing date values
n) **LABEL** : A fixed text value.
o) **SERIAL** : An ever incrementing value (To be used only once in one form, and serves as primary key of that data)
p) **MONTHYEAR**: Sometimes you do not want Date, but only month and year alone. This control would enable you to enter values in MM-YYYY format.

As you have seen different data types, most of the data is treated as Strings, except DATE, FILE, NUMERIC and SERIAL. A file can be attached using FILE, and SERIAL can be used for auto incrementing number. But note that a SERIAL can be used only once in one set of fields in the form, and will be used as the primary key to identify the record.

LIST is a set of values to be shown in a drop box. And the values should be set as the third parameter, data source. For example

[FIELD:TYPE:LIST:SALARY,PROFIT,RENT,OTHERS]

It directly compares with ROCOMBO, which is again a list of values, but this time, the list of values will come from a FORMDEF datastore, using a SQL. For example

[FIELD:STATUS:ROCOMBO:SELECT DISTINCT EXPNAME FROM EXPENSES]

COMBO is the same as ROCOMBO, except the fact that it becomes editable to add a new value then and there itself. However this value will not reflect when a new record is added again.

LABEL type is a just read only field with text values(For example, you might want to show an About Form with version, date etc"

Note: There are two inbuilt fields which the user must never use , USER, and LASTUPT which are respectively User name and Last update time stamp, applied to each of the records that a user edits. The developer is further advised not to use SQL keywords for field names, like FROM, TO, LIKE, SUM etc.

There can be as many forms as you want, exchanging data among themselves, in your application.

4) The last section which may or may not be required by your application is REPORT

Reports are of two types (Importable, Non importable) Importable reports allow all users to export the report and import. However if the data elements are not sufficient, your data may become corrupt or redundant.

So make sure that if you are making a report Importable (by adding USER,LASTUPT as last fields in the report), all required columns are present.
Now the question arises, where does it import to?
The answer is simple: to one of the data stores.

[REPORT:<report name>:<datastore name>:<SQL>]

So an importable report should select all the columns in a data store, except (SERIAL, FILE, PICTURE)

A form with all labels cannot be exported/imported and may result in crash.

Irrespective of the import option, no REPORT should use BLOBS in the SELECT query.
Any report can be exported to HTML,CSV, TAB separated files, as the case may be.

Important note: While generating reports, do not use FILE, PICTURE, RICHTEXT fields as they are binary in nature and cannot be displayed in reports. You can use these fields in editing on the form, but not in reports and export/import options.

Using PForms.dll
Hence forth form release 0.2.4, the forms process (compile, create form file pf1, read form file pf1) have been made into a library which you can use in your own works. It requires System.Data.Sqlite as a dependency. The first thing you need to have for creating a new form is a file with the definition (a .pfdf file for example in pournami). Steps involved in using the PForms is

1) Add reference to PForms.dll in your project
2) Create a reference to the class VitalLogic.Libraries.PForms.Initiator
3) Register handler for MessageSent event for the initiator
4) Assuming that you have a form definition file, call the function of the initiator, ProcessContent(filepath). If your compilation is successful, then appropriate message would be sent.
5) Call the method ReadForm(formname) to load your form contents and returns a MetaObject… which hierarchically contains FormDefinitions, Fields, Reports etc. You will need to process them to place them good on your output design of Form.

Pournami Handles this Meta object to create a nice form for you. But you may not like the monotonous nature of Pournami forms, so based on your taste and choice, you can crete and use your forms.

In Pournami 0.3.3, the forms aspect also identifies, the current PFORM version used (FVer:0.2.4) and current form data that is read (DVer: 0.2.2) . FVer 0.2.4 can read data from forms of 0.2.2 and up.

Optionally the user can identify his forms with a version number 0.1, 0.2 whenever his form design changes. That can be done using a [FORMDEF:About:ABOUT] element with fields that are labels and carry some initial values. For example

[FORMDEF:About:ABOUT
[FIELD:Version:LABEL:0.1]
[FIELD:Author:LABEL:Tom Alter]

Vijay Sridhara
06 Aug 2012