

Building a data form

In this example we will build a data form that inherits the frameCustomDbBtn. This frame class was build to developer data forms with speed and efficiency. Uses sqlalchemy to do automatically operations of Insert, Update and Delete in database. The developer just need to identify the fields (for fields in the ORM class) and let the pyHed, using SQLAlchemy, to do the heavy work.

–**params** : Any parameter or list of parameters that you want to send to the frame class;

–**allowedOperations**: A list that contains the allowed operations on the frame (insert, update or delete). By default, all operations is allowed.

1º Step – Creating table or view mapping in ORM class: In our sample application, we mapping the "Pessoa" table. Following the class.

```
#Get sequence
PESSOA_SEQ = sqlalchemy.Sequence("PESSOA_SEQ")
#Mapper table
class PESSOA(object):
    def __init__(self, ID_PESSOA=None, COD_PESSOA=None, NOME=None, SEXO=None,
                 DATA_NASCIMENTO=None, OBSERVACAO=None, ID_TIME_FUTEBOL=None):
        self.ID_PESSOA = Dbinst.getEngine().execute(PESSOA_SEQ)
        self.COD_PESSOA = self.ID_PESSOA
        self.NOME = NOME
        self.SEXO = SEXO
        self.DATA_NASCIMENTO = DATA_NASCIMENTO
        self.OBSERVACAO = OBSERVACAO
        self.ID_TIME_FUTEBOL = ID_TIME_FUTEBOL

pessoa_table = sqlalchemy.Table('PESSOA', Dbinst.getMetaData(),
                                sqlalchemy.Column('ID_PESSOA', sqlalchemy.Integer, primary_key=True),
                                sqlalchemy.Column('COD_PESSOA', sqlalchemy.Integer),
                                sqlalchemy.Column('NOME', sqlalchemy.String(100)),
                                sqlalchemy.Column('SEXO', sqlalchemy.String(1)),
                                sqlalchemy.Column('DATA_NASCIMENTO', sqlalchemy.Date),
                                sqlalchemy.Column('OBSERVACAO', sqlalchemy.String(255)),
                                sqlalchemy.Column('ID_TIME_FUTEBOL', sqlalchemy.Integer)
                                )
sqlalchemy.orm.mapper(PESSOA, pessoa_table)
```

This is the default mapping for ORM classes.

2º Step - Creating the form: To create a data form with the pyHed it's simple and fast. To do this, you need to create a file inside “ desktop ” folder (this directory structure it's not required, we just have this structure to make thing's more easy to understand). Our file will be named frameCadPessoa.py. Note that we are using a default nomenclature for this file, beginning with 'frame' indicating that contains inheritance, 'CadPessoa' indicating it's a person register form. It's not necessary follow this pattern, but, using this make its easier the manipulation.

On this moment we will just create a form and make it appear. Write in file the follow lines.

```

#-*- encoding: iso-8859-1 -*-
#-*- coding: iso-8859-1 -*-

# imports necessários
import sys
import PyQt4
import appConsts
sys.path.append(appConsts.appDbPath)
import appDbTables
from pyHed.frames import *
from pyHed.common import *
from pyHed.components import *

# Cria a classe herdando frameCustomDbBtn
class FrameCadPessoa(frameCustomBtnDb.FrameCustomBtnDb):
    # Inicialização da classe
    def __init__(self, parent, params=None):
        super(FrameCadPessoa, self).__init__(parent, params=params)

        ### |parâmetros de inicialização da tela ###

    # Método onPaint
    # Neste método são criados os componentes da tela
    def onPaint(self):
        super(FrameCadPessoa, self).onPaint()

        ##### componentes #####

```

Save this file.

Now, your form is ready to appear on the screen, but there are two others steps to become the frame visible. Add the new class on the application frameMain (On this example novaAppMain.py) and insert the action in database. Now we will add in frameMain.

Open the file and add frameCadPessoa in import list. Follows the example:

```

#-*- encoding: iso-8859-1 -*-
#-*- coding: iso-8859-1 -*-

import sys
import PyQt4
import appConsts

# import pyHed classes
from pyHed.common import *
from pyHed.forms import *
from pyHed.frames import *
from pyHed.components import *

# import application frames
import frameLogin

# Import dos Frames de Cadastro do Sistema
import frameCadPessoa
import frameGerenciaPessoas

```

Import is OK, now we will create the menu. See example below:

```
# Main class
class FrameNovaAppMain(frmCustomMain.FrmCustomMain):
    def __init__(self):
        # Login and start app
        super(FrameNovaAppMain, self).__init__(frameLogin.FrameLoginNovaApp,
                                                '%s/splashScreen.png' % appConsts.appImagePath,
                                                windowMaximize=False)

        super(FrameNovaAppMain, self).setIcon('%s/appIco.ico' % appConsts.appImagePath)

    def registerFrames(self):
        """ frames registration """
        # Cria o menu.
        menuCadastro = self.addMenu('&Cadastros')
        # Cria o Item de Menu e atribui a ação a self.actCadPessoa
        # self.addItem(nome do menu, Classe, 'Nome da tela')
        self.actCadPessoa = self.addItem(menuCadastro,
                                          frameCadPessoa.FrameCadPessoa,
                                          u'Cadastro de Pessoas')
```

Completing this part, let's insert the action. Before a note: Note that on create the menu item, it was attributed on self.actCadPessoa, this is means that the action also was attributed on this variable. Note that the main form in pyHed applications have three big icons, this icons are called "quick access", for each icon there are a event, see below:

evtBtnLeftClicked(self):

evtBtnCenterClicked(self):

evtBtnRigthClicked(self):

Isn't necessary to explain which event is assigned to each icon. Them for attribute the action to quick access, execute this two steps:

- On method onPaint() you need to define the image button and hint;
- On the corresponding button method execute the action.

Follows the example:

```
...
def onPaint(self):
    # Paint event
    super(FrameNovaAppMain, self).onPaint()

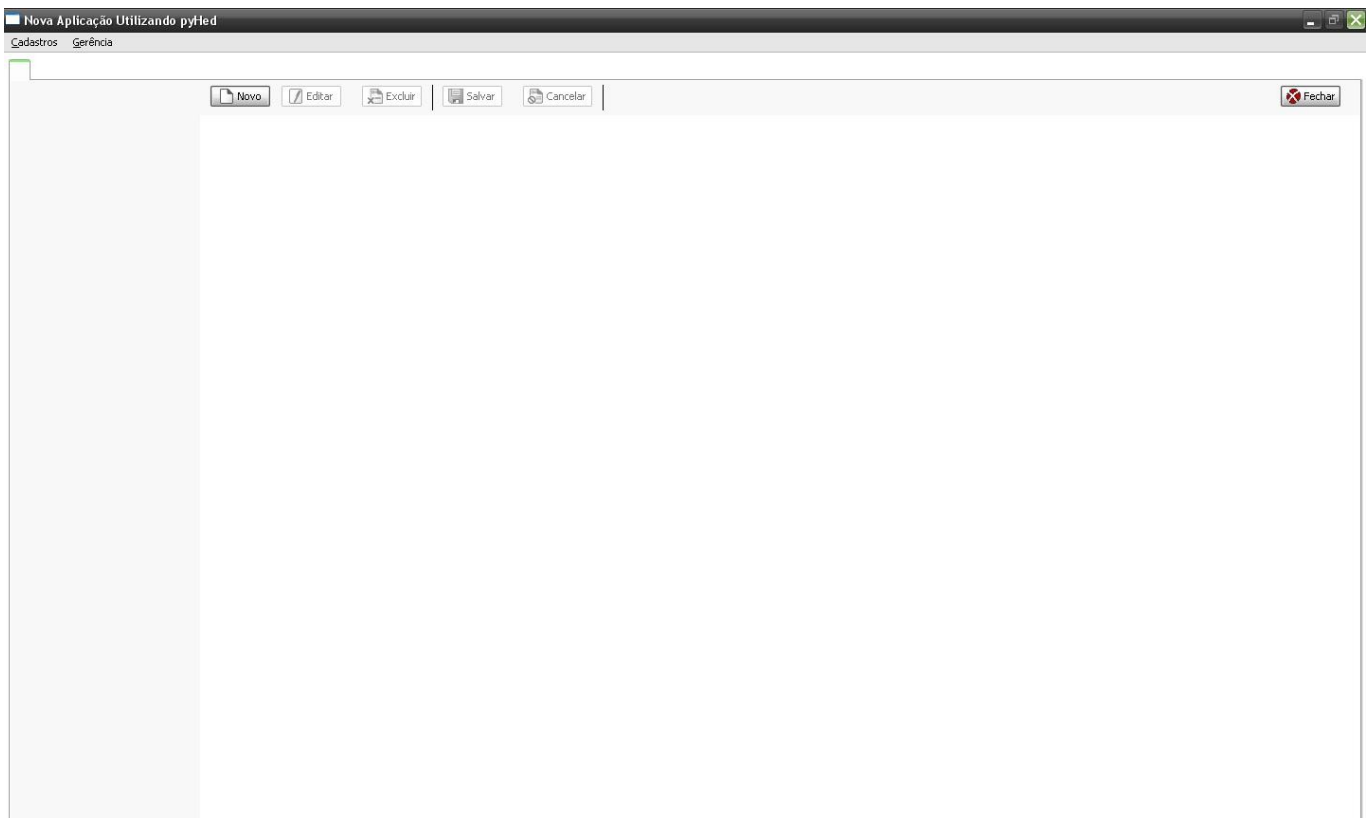
    self.btnLeft.setIcon(PyQt4.QtGui.QIcon('%s/medalha_cadastro_pessoa.png' % appConsts.appImagePath))
    self.btnLeft.setToolTip('Cadastro de Pessoa')
    ...
def evtBtnLeftClicked(self):
    self.actCadPessoa.trigger()
```

Now the last step: insert the action on database. This is the most simple part, to insert the action use the following insert:

```
insert into Acao (ID_Acao, ID_Acao_Pai, Nome, Caption,
                Data_Cadastro, Nro_Execucao, Arquivo, Atalho)
values (
    gen_id(Acao Seq, 1), --- ID da ação, buscar da sequence
    1, -- ID da ação pai, neste caso a ação ROOT
    'FRAMECADPESSOA', -- O nome do frame em caixa alta
    'Cadastro de Pessoas', -- Título para a tela.
    current_date, -- Data de cadastro da ação
    0, -- Nº de execuções
    null, -- Arquivo (Não utilizado neste exemplo)
    null) -- Atalho (Não utilizado neste exemplo)
```

This action is used by pyHed to control the user access on the application form's. We will make another help documentation talking particularly about this item.

OK ! Now, your form (frame class) will appear. Execute the application and access the form, it should appear like this:



Now that the form is visible, we finished the building of it.

Init parameters

To execute insert, update and delete actions and make automatically searches its necessary configure some parameters on the `__init__` method on your frame class. Follows the

example:

```

class FrameCadPessoa(frameCustomBtnDb.FrameCustomBtnDb):
    # Class init
    def __init__(self, parent, params=None):
        super(FrameCadPessoa, self).__init__(parent, params=params)

        # Set dbClass
        self.dbClass = appDbTables.PESSOA

        # Set title
        self.title = 'Cadastro de Pessoa'

        # Query to search
        self.SearchSQL = u"""
            select
                ID_PESSOA,
                COD_PESSOA,
                NOME
            from
                PESSOA
            where
                %WHERECLAUSE% and
                ID_PESSOA <> 1
        """

        # Set de coluns title
        # The colum that contains * before de name is the key of grid and will be hide.
        self.SearchColumnsTitle = ['*ID_PESSOA',u'Cód. Pessoa', u'Nome']

        # Set the search fields
        self.SearchFields.append('Nome,NOME,string')
        self.SearchFields.append(u'Cód Pessoa,COD_PESSOA,integer')

        #Set the primary key
        self.PrimaryKey = 'ID_PESSOA'

        if (params is not None):
            self.ID_PESSOA = params[1]
            self.filterRegister("ID_PESSOA=%s" % (self.ID_PESSOA))
        else:
            self.ID_PESSOA = ""
  
```

About the parameters:

– **self.dbClass** : Is the table or view mapping in ORM class. The first step on our example;

– **self.title**: the frame title;

– **self.SearchSQL**: This is the SQL that will be executed on the search frame class. It must contain the table Primary Key and its fields that you want show on the grid results. You MUST inform the clause %WHERECLAUSE% on the primary where. This substitution variable will be substituted on executing time by the where clause generated by the search frame class. *
 NOTE: on future, pyHed will allow the developer to user a ORM class instead of a sql string.

- **self.SearchColumnsTitle:** Is a list with fields titles on search grid results. Note that the first is the table primary key and is preceded by *, Its means that it will be a hide column and will contains the primary key necessary to open data form by search form. IMPORTANT: The order of columns should be same of sql, be careful with it for your form work.
- **self.SearchFields:** Is a list that contains the search filters. Each item should be a string separated by comma that should contain the follow order: "*label, FIELD, type*".
Where:
 - **label:** Text that is visible to user;
 - **FIELD:** The clause that will be inserted.
Obs.: If your select contain any join statement and your tables have any alias for example "PESSOA P" the string value should be P.NOME as the example above. Its avoid problems as ambiguous columns.
 - **type:** Is the field type.
 - Allowed types: string, integer, date;
- **self.PrimaryKey:** Must be informed the primary Key of table or view.

The code bellow is necessary **JUST** to open the frame class by another frame on the application. Using the parameter **params** with is the default parameter to send data, values, etc to another classes. We will see that work in example of create management frame.

```
if (params is not None):
    self.ID_PESSOA = params[1]
    self.filterRegister("ID_PESSOA=%s" % (self.ID_PESSOA))
else:
    self.ID_PESSOA = ""
```

Ready! The boot parameters are OK.

onPaint method

This method is used to create the frame class components. On this example we wil not detail these components, there are a specific document for it.

Example:

```

def onPaint(self):
    super(FrameCadPessoa, self).onPaint()
    self.hidePnlSubMenu()

    # Set the components
    # label
    self.lblCodPessoa = components.Label(self.PnlMain,
                                         text=u'Cód. Pessoa:',
                                         width=75, x=10, y=10)

    # DbLabel
    self.DBlblCodPessoa = dbComponents.DbLabel(self.PnlMain,
                                                dataField='COD_PESSOA',
                                                x=self.lblCodPessoa.x()+self.lblCodPessoa.width()+2,
                                                y=10, width=300,
                                                defaultStyle=False)

    # DbTextEdit
    self.edtNome = dbComponents.DbEdit(self.PnlMain,
                                       dataField='NOME',
                                       text=' &Nome:',
                                       x=10,
                                       y=self.lblCodPessoa.y()+30,
                                       width=350)

    # DbComboBox
    self.lkpSexo = dbComponents.DbComboBox(self.PnlMain,
                                           dataField='SEXO',
                                           items=['M', 'F'],
                                           text='&Sexo',
                                           width=100,
                                           x=10,
                                           y=self.edtNome.y()+30,
                                           required=True)

    # DbDateEdit
    self.edtDtNac = dbComponents.DbDateEdit(self.PnlMain,
                                             dataField='DATA_NASCIMENTO',
                                             text=u'Data Nascimento',
                                             width=100,
                                             x=self.edtNome.x()+self.edtNome.width()+20,
                                             y=self.edtNome.y()-21)

    # DbComboSearcher
    self.cbsTimeFutebol = dbComponents.DbComboSearcher(self.PnlMain,
                                                        dataField='ID_TIME_FUTEBOL',
                                                        searchEvent=self.evtPesquisaTime,
                                                        text=u'Time para qual torce',
                                                        size=(300,110), x= 10,
                                                        y=self.lkpSexo.y()+30,
                                                        hint=u'Selecione o clube para o qual a pessoa torce.')

    # DbMemo
    self.mmObs = dbComponents.DbMemo(self.PnlMain,
                                     dataField='OBSERVACAO',
                                     text=u'Observação',
                                     width=300,
                                     height=250,
                                     x=10,
                                     y=self.cbsTimeFutebol.y()+self.cbsTimeFutebol.height()+30)
    
```

Note that on DbComboSearcher an event is passed by parameter, As discussed previously we will not detail the components here, however you will need it to inform this method to your frame class works.

Follows example code;

```
def evtPesquisaTime(self, idValue=None):
    """
    Evento do botão de pesquisa do time.
    Este evento também engloba a exibição de dados na tela quando o usuário
    seleciona um registro para visualização.
    Caso o usuário estiver pesquisando um valor, o parâmetro idValue virá None.
    Caso seja um valor que precisa ser exibido na tela, o parâmetro trará
    o id da sql
    """
    - idValue = se não for None, retorna o id para EXIBIÇÃO de dados na tela.
    """
    baseSql = """
    select
    CF.ID_CLUBE as ID_CLUBE,
    CF.NOME as NOME
    from
    CLUBE_FUTEBOL CF
    where
    l=1
    """

    sql=''
    if idValue is None:
        if self.cbsTimeFutebol.getSearchText() == '':
            raise pyHedExceptions.UserException("É necessário informar um valor para pesquisa.", "Atenção")
        sql = baseSql+"""and (select * from NORMALIZE(upper(CF.NOME))) like
        |(select * from NORMALIZE(upper('%'+self.cbsTimeFutebol.getSearchText()+ '%'))|)"""
    else:
        sql = baseSql+" and CF.ID_CLUBE =" +str(idValue)

    rs = pyHedConsts.dbInst.getConnection().execute(sql).fetchall()

    if len(rs) > 0:
        self.cbsTimeFutebol.setResult(rs)
    else:
        self.cbsTimeFutebol.setResult(rs)
        raise pyHedExceptions.UserException(u'A pesquisa não retornou dados.')
```

Your frame class it's ready! Should appear to you like this:

Data Form:

Cadastro de Pessoa

Novo | Editar | Excluir | Salvar | Cancelar | Pesquisar

Fechar

Nome: Data Nascimento: 11

Sexo:

Time para qual torce: Valor para pesquisa: Resultado:

Observação:

Search Form

Pesquisar: Cadastro de Pessoa

Adicionar condição | Remover condição | Pesquisar

Fechar

Campo: Nome Operação: Contendo Condição: a|

Filtros informados

Exibindo todos os registros

	Cód. Pessoa	Nome
1	789	Vinicius Marconi
2	27	João da Silva Gulari
3	28	Rodrigo Jesuino da Silva
4	29	José da Silva

Congratulations! You build your first form! Stay tuned to others documentations...