

Construindo uma tela de cadastro

Neste exemplo vamos construir uma tela de cadastro que herdará o `frameCustomDbBtn`. Este frame foi construído para desenvolver telas de cadastro com rapidez e eficiência. Utiliza o ORM do `sqlAlchemy` para fazer as operações de Insert, Update e Delete automaticamente no banco de dados, cabendo ao usuário somente identificar os campos (em relação aos campos da classe no ORM) e deixar que o `pyHed`, via `sqlAlchemy`, cuide do "pesado".

Seus parâmetros na inicialização são:

- **params** : qualquer parâmetro ou lista de parâmetros que se queira passar para uma tela;
- **allowedOperations**: uma lista com quais operações são permitidas na tela (insert, update ou delete). Por default, TODAS são permitidas.

1º Passo – Criando o mapeamento da tabela ou view na classe do ORM: No nosso exemplo de aplicação mapeamos a tabela "Pessoa". Segue a classe.

```
#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)
```

Este é o padrão a ser seguido no mapeamento de classes do ORM.

2º Passo - Criando a tela: Criar uma tela de cadastro com o `pyHed` é uma tarefa simples e rápida. Crie um arquivo dentro da pasta desktop, vamos chamar o nosso arquivo de `frameCadPessoa.py`. Note que estamos usando uma nomenclatura padrão para o nome do arquivo, começando com 'frame' o que indica que ele possui uma herança, 'CadPessoa' indicando que é uma tela de cadastro de pessoas. Não é necessário que você faça desta forma porém utilizando este padrão torna mais fácil a manipulação dos arquivos. Neste primeiro momento vamos somente criar a tela e fazê-la aparecer. Insira as seguintes linhas no seu arquivo:

```

#-*- 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 #####

```

Salve o arquivo.

Agora sua tela já está pronta para “aparecer”, porém ainda faltam 2 passos para que ela se torne visível, adicioná-la ao frameMain da aplicação (no nosso exemplo novaAppMain.py) e inserir a ação no banco. Primeiro vamos adicioná-la ao frameMain.

Abra o arquivo e adicione o frameCadPessoa.py na lista de imports segue exemplo:

```

#-*- 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 realizado, vamos a criação do menu. Veja o exemplo abaixo:

```
# 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')
```

Feito isto, agora só falta inserir a ação. Mas antes uma observação: Note que na criação do item de menu ele foi atribuído a `self.actCadPessoa`, isto significa que a ação também foi atribuída a esta variável. Note que na tela inicial das aplicações baseadas no pyHed há três imagens grandes, estas imagens são para acesso rápido a uma ação, para cada imagem há um evento pré definido, são eles:

`evtBtnLeftClicked(self):`

`evtBtnCenterClicked(self):`

`evtBtnRigthClicked(self):`

Não é necessário dizer a qual botão cada um é atribuído. Então para atribuir a ação de cadastro de pessoas a um atalho rápido basta dois passos:

- No método `onPaint()` defina a imagem do botão e o hint.
- No método correspondente ao botão execute a ação.

Segue o exemplo:

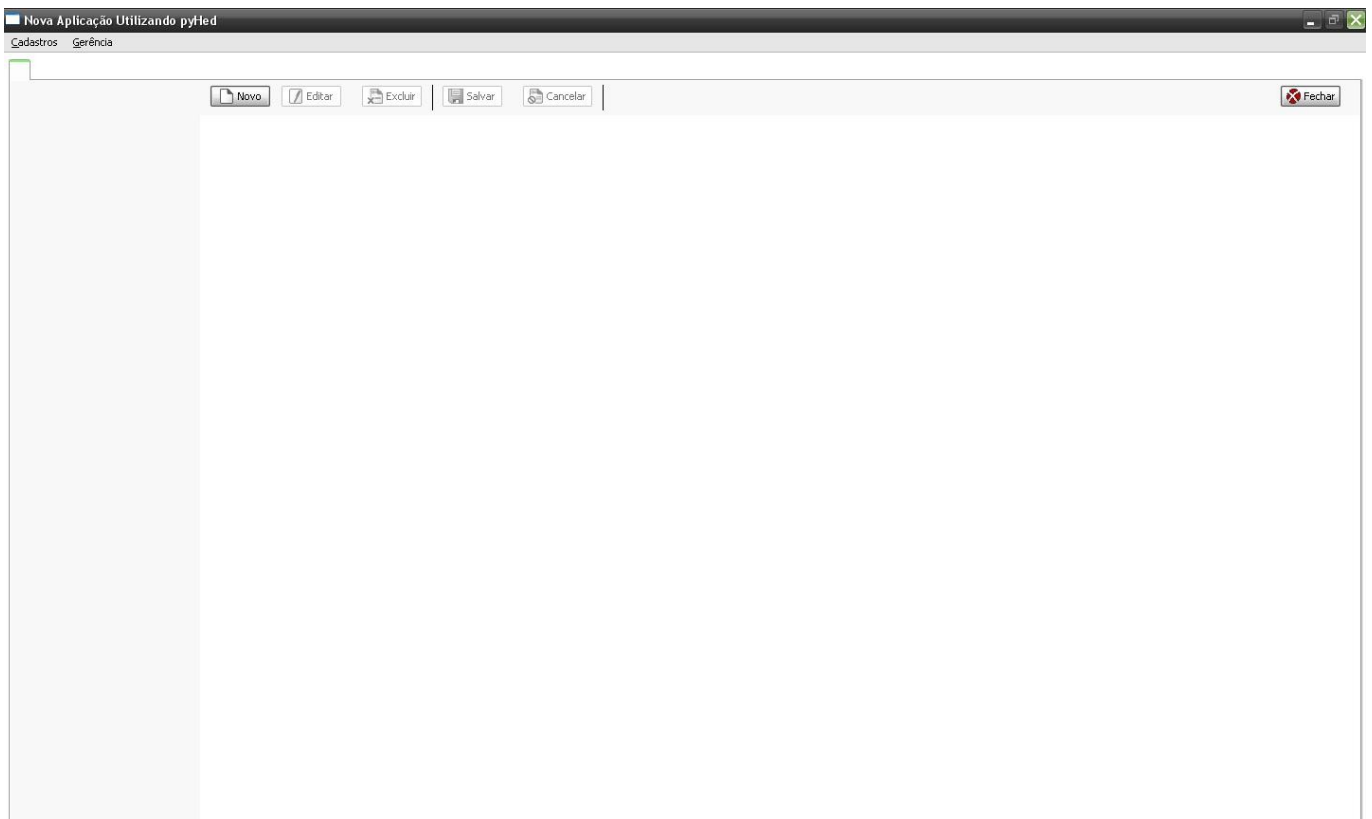
```
def onPaint(self):
    # Paint event
    super(FrameNovaAppMain, self).onPaint()

    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()
```

Agora o ultimo passo, inserir a ação. Esta é a parte mais simples, para inserir a ação use o seguinte 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)
```

Pronto ! Sua tela irá aparecer. Execute a aplicação e acesse a tela, deverá aparecer desta forma:



Agora que a tela está visível, vamos de fato construir-la.

Parâmetros de inicialização

Para que sua tela execute as ações de insert, update e delete e faça buscas automaticamente, é necessário configurar alguns parâmetros no método `__init__`. Segue um exemplo de código:

```

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 = ""
  
```

Agora vamos conhecer parâmetro a parâmetro:

- **self.dbClass** : É a tabela ou view mapeada na classe do ORM. O primeiro passo do nosso exemplo;
- **self.title**: Título do frame;
- **self.SearchSQL**: Esta é a sql que será executada na pesquisa da tela, sempre deve conter a Primary Key da tabela e os campos que deseja mostrar no grid de resultados. SEMPRE deve haver a clausula %WHERECLAUSE% ela será substituída em tempo de execução pelas cláusulas criadas pelo usuário.
Obs.: No futuro o pyHed permitirá o programador informar a classe mapeada no ORM ao invés de uma SQL.

- **self.SearchColumnsTitle:** É uma lista com os títulos das colunas no grid de resultados da pesquisa. Note que o primeiro é a chave primária da tabela antecedida por um *, isto significa que ela será uma coluna escondida que conterá a chave primária para passar para a tela de cadastro quando houver a seleção. Note que a ordem das colunas na lista é a mesma em que os campos se dispõem na select de pesquisa, tome sempre o cuidado de manter esta ordem, caso contrário não funcionará como esperado.
- **self.SearchFields:** É uma lista com os filtros que o usuário poderá utilizar para realizar a pesquisa. Cada item deve ser uma string separada por virgulas que deve conter na seguinte ordem: *“label, CAMPO, tipo”*. Onde:
 - **label:** Texto que será visível ao usuário;
 - **campo:** a clausula que será inserida
Obs.: se a sua select possuir algum 'join' e as tabelas possuírem alias por exemplo PESSOA P o valor na string deve ficar P.NOME seguindo o exemplo acima. Isto evita problemas com colunas ambíguas;
 - **tipo:** É o tipo do campo.
 - Tipos permitidos: string, integer, date;
- **self.PrimaryKey:** Deve ser informado a chave primária da tabela ou view em questão.

O trecho de código abaixo é necessário **SOMENTE** para abrir a tela a partir de outra como uma tela de gerência por exemplo, utilizando o parâmetro params é passada a chave primária para montar a tela com um registro ativo. Veremos melhor como funciona no exemplo de criação de uma tela de gerência. Fica o código para conhecimento.

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

Pronto! Os parâmetros de inicialização da classe estão completos.

Método onPaint(self)

É neste método que criamos os componentes da tela. Neste exemplo não detalharemos os componentes pois haverá um documento específico para isto.

Segue o exemplo de código:


```

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 que no componente DbComboSearcher é passado um evento por parâmetro, como falado anteriormente não detalharemos o funcionamento dos componentes haverá um documento para isso, no entanto você vai precisar deste evento para que sua tela funcione. Segue o código como exmplo:

```
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.')
```

Feito isto sua tela está pronta ! Deverá aparecer para você desta forma:

Tela de cadastro.

Novo Aplicação Utilizando pyHed

Cadastros Gerência

Cadastro de Pessoa

Novo Editar Excluir Salvar Cancelar Pesquisar Fechar

Cód. Pessoa:

Nome: Data Nascimento: 11

Sexo:

Time para qual torce:

Valor para pesquisa:

Resultado:

Observação:

Tela de pesquisa.

Novo Aplicação Utilizando pyHed

Cadastros Gerência

Cadastro de Pessoa Pesquisar: Cadastro de Pessoa

Adicionar condição Remover condição Pesquisar Fechar

Campo: Nome Operação: Contendo Condição: a|

Filtros informados:

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

Exibindo todos os registros

Parabéns! Você criou sua primeira tela.