Padrões de programação SCI

De WikiSCI

1) PADRÕES DE PROGRAMAÇÃO (PADRÕES DE CODIFICAÇÃO E DE LÓGICA)

Objetivo: Muitas são as formas de se fazerem uma rotina, função ou procedure, fazendo com que elas cheguem ao mesmo resultado. Porém é sempre melhor utilizarmos a maneira mais clara, rápida de entender, com menos código fonte possível, reaproveitando ao máximo rotinas já feitas. Com isso, construímos a rotina em menos tempo e ela se tornará mais segura, pois será feita dentro de um método padrão, utilizado em outras rotinas já testadas e aprovadas. Para programar dessa maneira, é preciso trabalhar dentro de padrões de programação. Padrões de CODIFICAÇÃO e de LÓGICA. Os padrões de codificação visam melhorar a leitura e interpretação do código-fonte. Envolvem a escolha dos nomes de variáveis, objetos, quais comandos utilizar, quando utilizar letras maiúsculas ou minúsculas, identação, etc. Já padrões de lógica visam maximizar o desempenho das rotinas, deixando-as mais fáceis de entender e de dar manutenção, mais rápidas de programar, mais seguras, enfim, com um resultado final mais próximo da perfeição possível.

Existem algumas regras de padronização que parecem (e são) muito chatas, principalmente quando se referem a que comando utilizar, que identação utilizar, onde e quando utilizar letras maiúsculas ou minúsculas, etc. Porém, o objetivo da padronização é fazer com que todos os programas parecam ser escritos pelo mesmo programador, ou seja, valorizar o sentido do trabalho em grupo. Para isso ser possível, todo mínimo detalhe deve ser observado afim de adaptar o programador da maneira mais veloz possível, sendo que em pouco tempo o programador estará escrevendo seus programas no formato padrão sem nem mesmo perceber. A individualidade na programação deve ser valorizada também, e muito. Senão, teremos programadores trabalhando como robôs dentro de um padrão, sem capacidade para entender, analisar e discutir maneiras de se melhorar a forma de trabalho. Porém, toda idéia, sugestão ou crítica deve ser canalizada para a discussão em grupo, e o tempo a ser perdido com a análise deve ser proporcional a importância e ao retorno da idéia sugerida.

Por fim, o modelo de padronização apresentado a seguir foi desenvolvido pelos técnicos da Santa Catarina Informática após minuciosos estudos e simulação de inúmeras ocorrências, para que se tratasse cada situação da melhor maneira possível, não esquecendo das conseqüências que isso causarão ao conjunto dos sistemas. Alguns sistemas da Santa Catarina Informática não foram desenvolvidos totalmente dentro desses padrões, devido ao fato de terem sido desenvolvidos em paralelo aos projetos padrões. Lembramos mais uma vez que os padrões podem ser aumentados ou alterados, sempre com análise prévia, minuciosa e supervisionada. Todos os textos abaixo estarão exemplificados através dos comandos e funções do DELPHI, que é a linguagem de programação utilizada pela Santa Catarina Informática.

1.1) PADRÕES DE CODIFICAÇÃO

Sem padrões de codificação, será mais difícil você compreender o código-fonte de outro programador. Até mesmo o criador da rotina poderá ter dificuldades em compreendê-la se não trabalhar com padrões. Abaixo algumas dificuldades em se trabalhar sem padrões de codificação:

- Identificar a seqüência da rotina, seja dentro de condições, loops ou de uma rotina direta. - Identificar a que se refere determinada informação: se é uma variável, se é um objeto, se é função, etc. - Perda de tempo para decidir entre uma forma ou outra de se fazer a mesma rotina. - Visualização ruim. - Em geral, difícil manutenção das rotinas.

REGRAS DE CODIFICAÇÃO:

1.1.1) IDENTAÇÃO:

Não colocar comandos na linha do BEGIN; sempre começar o bloco uma linha abaixo 2 colunas à direita O END deve estar na mesma coluna do BEGIN. Em caso de IF, colocar o comando da condição verdadeira na linha abaixo com 3 colunas à direita. O ELSE deve estar na mesma coluna do IF.

ex:

procedure pTeste

begin

```
pProcesso1;
```

```
if wVerdadeiro then
   begin
     pExecuta1;
     pExecuta2;
   end
else
   begin
     pExecuta2;
     pExecuta1;
   end;

pProcesso2;
```

end;

No caso de While (loops), o comando subseqüente deve estar na linha de baixo 2 colunas à direita.

Ex:

begin

```
wContagem := 1;
```

```
while wContagem <= 10 do
  begin
```

```
    pProcesso1;
    pProcesso2;

    if fCondicao1 then
       pProcesso3;
```

```
    if fCondicao2 then
       begin
         pProcesso4;
         pProcesso5;
       end
    else
       begin
         pProcesso6;
         pProcesso7;
       end;
```

```
    wContagem := wContagem + 1;
```

```
  end;
```

end;

Procure deixar uma linha em branco abaixo e acima da maioria dos WHILEs e IFs que você tiver no seu programa. Não se deve acrescentar as linhas em branco apenas quando a rotina se tornar muito cheia de condições e loops, porque nesse caso o programa ficará com uma aparência muito vazia e confusa.

1.1.2) COMANDOS PREFERENCIAIS:

Existem vários comandos diferentes que na verdade fazem a mesma coisa. Exemplo disso é WHILE, REPEAT e FOR-NEXT. Devemos optar por um deles para trabalhar nos sistemas, porque se você não está acostumado ao REPEAT por exemplo, e de repente você vê o REPEAT em alguma rotina, você terá que perder tempo para aprender ou relembrar como é o funcionamento do REPEAT. Por isso, no caso de rotinas de loops, deve-se utilizar sempre e somente o WHILE. No caso de testes de condição deve-se utilizar o IF ao invés do CASE, salvo em condições onde haverá múltipla-escolha. Quando utilizar AND ou OR, deve-se colocar cada condição entre parênteses. Ex:

if (not wTeste > 10) and (wTeste2 < 20) then ...

Obs: Evite utilizar comandos que lhe pareçam pouco usados. Pesquise em outros programas com rotinas parecidas e tome base para saber qual o comando ideal.

1.1.3) UTILIZAÇÃO DE LETRAS MAIÚSCULAS E MINÚSCULAS NO CASO DE COMANDOS E FUNÇÕES DO DELPHI:

Quando um comando ou função for formado por uma palavra apenas, deve ser escrito em minúsculo. Ex: begin, while, copy. Quando um comando, função ou constante do Delphi é composto por mais de uma palavra, deve ser escrito em minúsculo com a primeira letra de cada palavra em maiúsculo. Ex: AnsiUpperCase, CopyFile, aviCopyFile. Note que no último exemplo (aviCopyFile), existe um prefixo (avi). Nesse caso se escreve o prefixo em minúsculo (já que não é uma palavra) e o restante em minúsculo com a primeira letra de cada palavra em maiúsculo.

1.1.4) UTILIZAÇÃO DE LETRAS MAIÚSCULAS E MINÚSCULAS NO CASO DE COMANDOS E FUNÇÕES PRÓPRIAS:

As regras são as mesmas do item 1.1.3. Apenas há mudanças para objetos TTable e referências a nomes de campos de tabelas. Quando for TTable, o objeto deve ter o prefixo 'tb' mais o nome da tabela em maiúsculo. O mesmo para campos de tabela, porém o prefixo será 'bd'. Ex: tbLANbdCONTA.AsInteger. Deve-se observar sempre a existência de prefixo antes de funções, procedures, variáveis e objetos. Sobre os prefixos a serem usados veremos mais adiante. Ao criar funções, variáveis, etc. é muito importante dar nomes bastante expressivos e não se preocupar muito com o tamanho dos nomes. Pode-se utilizar nome longos, desde que o espaço utilizado seja bem aproveitado e expressivo. Apenas quando os nomes passarem a ser exageradamente grandes deve-se recorrer a abreviações.

1.1.5) COMENTÁRIOS:

É muito importante acrescentar comentários às rotinas. Especialmente quando a rotina tiver alguma característica diferente das outras, ou quando for complexa. Eles ajudarão a compreender melhor o objetivo dos passos tomados em cada ponto da rotina. Parece ser chato e perda de tempo colocar comentários nas rotinas, porém eles economizam valioso tempo no futuro. Você depois também terá certeza absoluta do que estiver fazendo. Desenvolva primeiro a rotina e depois insira comentários nela, para que isso não atrapalhe você no desenvolvimento da rotina.

1.1.6) REGRAS PARA DAR NOME A VARIÁVEIS, OBJETOS, FUNÇÕES, PROCEDURES, ETC:

Abaixo segue uma lista com as regras e exemplos de como nomear os dados que você irá trabalhar:

Tipo de Dados

Variáveis

Tipos: Local Global Privada Protegida Pública Publicada

Prefixos / Regras:

Local, Privada, Protegida: A primeira letra da variável obrigatoriamente deve ser “w” minúsculo. ex.: wNome_Variavel: Tipo_Variável;

Global: Não é aconselhável criar variáveis deste tipo. ex.: wpNome_Variável: Tipo_Variável;

Pública: Não deve ser criado variáveis deste tipo.

Publicada: Deve ser criado com o nome elegante. ex.: property Tabela: TTable read wTabela write wTabela;

Definições de tipos de variáveis

Local: é definido no escopo de um “procedimento ou função”. ex. *procedure pSoma;* var

```
wLocal: String;
```

begin

```
…
```

end;

Global: é definido no escopo da unidade principal. ex. *interface*

uses

```
Classes, ExtCtrls, Graphics;
```

var

```
wpGlobal: String;
```

implementation Privada, Protegida, Pública, Publicada: ex. t*ype*

```
TVariaveis = class
private
   wPrivada: Byte;
protected
   wProtegida: Integer;
public
   wPublica: String;
published
   property Tabela: TTable read wTabela write wTabela;
end;
```

Definição de:

Procedimentos: ex. *procedure pNome_Procedure;*

Função: ex. *function fNome_Funcao: String;*

Parâmetros de Procedimento e Função: ex. *procedure pNome_Procedure(prParâmetro: String);*

function fNome_Funcao(prParâmetro: String): Integer;

Tabelas:

Nome do Arquivo no HD: - Deve ter no máximo 8 caracters; - Obrigatoriamente iniciar com "t" minúsculo; ex. "tNome_Tabela"

Nome do Arquivo no Projeto: - Tabela deve ser colocada no Modulo de Dados "Data Module"; - Tabela no Modulo de Dados de iniciar com "tbT"; ex. "tbTNome_Tabela"

Nome do Campo na Tabela: - Sempre utilizar nomes que defina bem as características do campo; - Obrigatoriamente iniciar com "bd"; ex. "bdNome_Campo" - Deve ser preenchido no Campo o "DisplayName";

Objetos Visuais:

Formulário: - Nome do Formulário de iniciar com "fr"; ex. frNome_Formulário

Menu: - Nome dos campos do menu deve iniciar com "mm"; ex. "mmNome_Campo"

PopUp: - Nome dos campos do PopUp deve iniciar com "pp"; ex. "ppNome_Campo"

Edit: - Nome do componente deve iniciar com "ed"; ex. "edNome_Edit"

Label: - Nome do componente deve iniciar com "lb"; ex. "lbNome_Label"

Button / BitBnt: - Nome do componente deve iniciar com "bt"; ex. "btNome_Botão"

SpeedButtons: - Nome do componente deve iniciar com "sp"; ex. "spNome_SpeedButtons"

Panel: - Nome do componente deve iniciar com "pn"; ex. "pnNome_Panel"

ListBox: - Nome do componente deve iniciar com "lb"; ex. "lb_Nome_ListBox"

Grids: - Nome do componente deve iniciar com "gr"; ex. "grNome_Grid"

ComboBox: - Nome do componente deve iniciar com "cb"; ex. "cbNome_ComboBox"

CheckBox: - Nome do componente deve iniciar com "ck"; ex. "ckNome_CheckBox"

GroupBox: - Nome do componente deve iniciar com "gb"; ex. "gbNome_GroupBox"

RadioButton: - Nome do componente deve iniciar com "rb"; ex. "rbNome_RadioButton"

RadioGroup: - Nome do componente deve iniciar com "rg"; ex. "rbNome_RadioGroup"

Image: - Nome do componente deve iniciar com "im"; ex. "imNome_Image"

Table: - Nome do componente deve iniciar com "im"; ex. "tbNome_Tabela"

DataSource: - Nome do componente deve iniciar com "ds"; ex. "dsNome_DataSource"

DataModule: - Nome do componente deve iniciar com “dm”; ex. “dmNome_DataModule”

Obs: Se você se deparar com algum tipo de objeto que não esteja na lista acima, utilize a melhor abreviação possível em 2 caracteres. No máximo 3 caracteres.

1.2) PADRÕES DE LÓGICA:

Vimos que os padrões de codificação servem para chegarmos mais rapidamente a um entendimento das rotina através de uma visualização agradável. Já os padrões de lógica além de também ajudarem na visualização e entendimento, irão influenciar na segurança do programa, já que utilizando padrões de lógica, a chance de um programa ter problemas diminui muito, já que ele estará rodando sob rotinas parecidas, já testadas e aprovadas. Os padrões de lógica influenciam na maneira de se programar, que conceitos utilizar, qual lógica preferir, etc. Em resumo, nossos padrões de lógica zelam ao máximo pela orientação a objetos e pelo reaproveitamento e compartilhamento de rotinas já criadas. Em programação, quanto menos se programa (digitar código-fonte) menos erros ocorrerão no seu programa.

1.2.1) EVITE CRIAR VARIÁVEIS PÚBLICAS:

As variáveis públicas, como o próprio nome já diz, são variáveis que serão acessadas por todas units do sistema. Deve-se evitar utilizá-las, dando preferência a utilização de parâmetros para procedures e funções. Se uma procedure ou função depender de alguma variável externa a ela, muitas vezes você pode esquecer de atualizar o valor da variável pública de acordo com a ocasião. Se a função ou procedure utilizar parâmetros, estes serão obrigatórios, e deverão ser tratados no momento da chamada do processo, exigindo maior análise e atenção do programador, evitando muitos erros. Só utilize variáveis públicas quando tiver pleno domínio dos locais onde o conteúdo delas será alterado. Mesmo assim só é interessante utilizar variáveis públicas no caso de informações que serão carregadas na entrada do sistema e permanecerão com o mesmo valor durante toda execução. Além disso, o acesso a variáveis públicas dentro de funções e procedures pode bloquear ou dificultar o compartilhamento do processo entre vários sistemas e pode gastar mais memória do que o necessário.

1.2.2) NÃO TRABALHE COM DUPLICAÇÃO DE DADOS:

Um erro muito comum entre os programadores é achar que deve-se "economizar" pesquisas nos bancos de dados. Achando que estas pesquisas serão demoradas, são criadas variáveis que receberão dados, aproveitando-se uma outra pesquisa já existente para outra finalidade. Exemplo disso : imagine uma tela de relatórios que irá imprimir dados de uma empresa. O primeiro dado solicitado na tela é a empresa que os dados serão impressos. Alguns programadores já salvam todos os dados da empresa em variáveis para que no momento da impressão não tenham que pesquisar novamente. Isso pode causar confusões na rotina, além de que no momento da impressão os dados podem estar desatualizados (sempre lembre que nossas aplicações permitem acessar várias telas, uma sobre as outras). Como regra geral, não há necessidade de colocar o mesmo dado em dois lugares diferentes, a não ser que o ganho na execução da rotina seja marcante. Deve-se lembrar que a pesquisa em banco de dados através de comandos como FindKey é sempre instantânea, e em determinadas rotinas nem é necessária, pois muitas vezes o registro desejado já é o registro atual da tabela.

1.2.3) PROCURE CRIAR O MÁXIMO DE FUNCTIONS E PROCEDURES POSSÍVEIS:

Toda rotina desenvolvida deve ser elevada ao máximo de re-utilização possível, pensando inclusive em adaptação para utilização em outros sistemas. Óbvio que isso não haverá necessidade de se fazer em uma rotina mínima e específica. Porém, as maiores e mais importantes rotinas do sistema devem sofrer uma análise cautelosa para verificar se existe possibilidade de padronizá-las em um nível mais alto. Ao padronizar uma função ou procedure para ser usada por todas as units de seu sistema, ela deve ser incluída em uma unit com nome = 'uUtil'+nome do sistema. Se for padronizada em nível de todos os sistemas, deve ser incluída na unit Util.pas (C:\DW\SCI\COMPONENTES), com prévia supervisão. Portanto, a regra é : Sempre procurar fazer uma rotina somente uma vez. Por exemplo : Se for fazer uma rotina para excluir todos os arquivos com extensão 'db' do diretório 'C:\SCI\VSUC', crie um parâmetro na rotina para se passar o diretório. Dessa forma a rotina servirá não só para o diretório 'C:\SCI\VSUC', e sim para qualquer outro diretório.

1.2.4) SE FOI CRIADA UMA FUNÇÃO PRÓPRIA PARA UTILIZAR NO LUGAR DE ALGUMA FUNÇÃO DO DELPHI, NÃO UTILIZE MAIS A FUNÇÃO DO DELPHI:

Exemplos disso é funções fStrToInt, que deve ser usada no lugar de StrToInt. Só é necessário criar funções cópias das que já existem no caso da função existente não atender completamente as nossas necessidades. No caso do StrToInt, por exemplo, havia o seguinte problema : quando uma string vazia (*) era convertida para numérico, ocorria problemas de conversão. A fStrToInt funciona exatamente igual a StrToInt, com a vantagem de que quando a string é nula, o resultado retornado é igual a zero. Portanto sempre deve-se utilizar fStrToInt, e não StrToInt, assim como deve-se utilizar qualquer função própria desenvolvida, ao invés de usar a função original do Delphi.*

1.2.5) MONTAGEM DE WHILES OU LOOPS: Ao criar-se um While ou loop, deve-se procurar seguir os padrões aqui descritos, para facilitar uma futura compreensão da rotina : 1. Quando o loop estiver contando, a variável contadora deve sempre começar com 1, jamais com -1 ou zero. 2. Na condição do loop deve sempre ser utilizado o termo de comparação <= (no caso de contagem progressiva). 3. Os desvios de loop devem ficar exatamente abaixo do While e se necessário for, com incrementação. 4. A incrementação deve ocorrer no final do loop, evitando-se comandos abaixo da incrementação. 5. No caso de leitura de alguma tabela ou qualquer lista de ítens, tendo ocorrido a eliminação do registro que se estava lendo no loop atual, não deve haver incrementação, pois senão o registro seguinte ao registro eliminado será ignorado, já que quando se elimina um registro, o registro seguinte passa a ser o registro atual. Exemplo de um loop correto, dentro dos padrões:

begin

```
wConta := 1;                                <---- wConta começando em 1.

while wConta <= prFinal do                  <---- Na condição do loop se utiliza <=.
  begin
```

```
    if wConta = (prFinal div 2) then
       begin
         wConta := wConta + 1;              <---- Desvio com incrementação.
         Continue;
       end;
```

```
    if wConta = prDeveParar then
       break;                               <---- Quebra de loop.
```

```
    lbContagem.Caption := fIntToStr(wConta);    <---- Processo ao qual o loop se destina.
```

```
    wConta := wConta + 1;                   <---- Incrementação do loop.
```

```
  end;
```

end;

1.2.6) ORIENTAÇÃO A OBJETOS - CONCEITOS

Tópicos Avançados – Delphi 6 página de 19 à 63.

1.2.7) DATAMODULES:

Todo sistema deverá ter um (ou mais, depende a necessidade) Data Module, que é um objeto onde se armazenam outros objetos, normal mente não-visuais, por questões organizacionais. Nós utilizaremos o Data-Module para guardar os objetos TTable e DataSource, que em um sistema serão numerosos e acessados em diversos locais, daí a importância de se ter um local único para eles. Pode-se criar mais de um Datamodule numa aplicação para separar as tabelas em grupos distintos, Ex: um datamodule para tabelas do sistema, outro para tabela de integração com outro sistema.

Importante: o Datamodule deve estar na lista "forms auto-create" do menu Project | Options do Delphi, para que seja criado no início da execução do sistema.

1.2.8) UTILIZAR O TTABLE OU TQUERY:

Para acessar tabelas deve-se sempre dar preferência TTable que estão nos DataModules dos sistemas, pois utilizando o TTable você vai utilizar as tabelas ja abertas em memória, evitando assim o trabalho de se buscar as informações no HD do servidor.
Mas existem casos que o componente TQuery é mais apropriado, principalmente quando é necessário puxar várias informações de tabelas diferentes. Mas é importante ressaltar que a query deve ser utilizado somente um comando sql deve resolver todo o problema, porque comandos sql em sequência são muito custosos para serem processados.
Na dúvida é recomendado utilizar os dois métodos e testar em rede para ver qual é mais eficiente para a situação do momento.

1.2.9) UTILIZAÇÃO DE CAMPOS-OBJETO:

Para acessar conteúdos de campos de arquivos, acesse os campos através de objetos. Ao acessarmos campos através de objetos, na própria compilação já nos são apresentados possíveis erros de digitação e acesso errôneo ao nome dos campos. O controle fica parecido portanto com o das variáveis. Portanto, não utilize FieldByName.

Crie campos objetos. Faça isso dando duplo-clique sobre o objeto TTable.

1.2.10) AO GRAVAR CAMPOS QUE SÃO COMBO-BOX, GRAVAR ITEMINDEX AO INVÉS DO TEXT:

Quando for gravar dados que são editados na tela através de objetos Combo-Box, grave a propriedade ItemIndex ao invés de gravar o conteúdo da propriedade Text do Combo. A NÃO SER que você tenha um ComboBox que tem um conteúdo que pode variar, então nesse caso pode ser mais seguro gravar a propriedade Text. Analise bem antes de cada caso. No FormCad, no pCarregaGravaAut é gravado/carregado ItemIndex.

1.2.11) CRIE OS EDITS NAS TELAS COM O MESMO NOME DOS CAMPOS DOS ARQUIVOS:

Se você tiver um campo chamado 'bdENDERECO', ao criar o componente de edição para esse campo, utilize o nome 'edENDERECO'. Com isso, você poderá utilizar funções e procedures prontas existentes nos formulários padrões que irão fazer a gravação dos campos e o preenchimento dos edits de maneira automática.

1.2.12) AO MEXER EM UNITS QUE SÃO PADRÕES PARA TODOS OS SISTEMAS:

É importante ter muita certeza da necessidade de alterar units padrões, já que estas atendem todos os sistemas. Quando precisar mexer em alguma unit que seja padrão para todos os sistemas, você deve comunicar a seu responsável, e será feita uma análise para ver a viabilidade da alteração. Sendo aprovada, o responsável vai liberar a manipulação da unit no seu micro, você fará a alteração, e depois de concluída você vai avisar o responsável para que ele verifique o que foi feito e devolva a unit para o repositório padrão.

Eis a lista de diretórios e suas funções:

DW DIRETÓRIO GERAL DE DESENVOLVIMENTO DOS SISTEMAS VISUAIS DW\SUCESSOR DIRETÓRIO DOS FONTES DO SUCESSOR DW\PRACTICE DIRETÓRIO DOS FONTES DO PRACTICE DW\SUPREMA DIRETÓRIO DOS FONTES DO SUPREMA

…

DW\SCI TODOS OS FONTES PADRÕES DA SANTA CATARINA INFORMÁTICA DW\SCI\COMPONENTES FONTES REFERENTES A COMPONENTES PADRÕES ENTRE OS SISTEMAS DW\SCI\REPOSITORIO CONTÉM FORMS (TELAS) PADRÕES ENTRE OS SISTEMAS DW\SCI\REPOSITORIO\FORMCAD FORM PADRÃO DE CADASTROS DW\SCI\REPOSITORIO\FORMDIG FORM PADRÃO DE DIGITAÇÃO EM ARQUIVOS SELECIONÁVEIS, MAS COM ESTRUTURA IGUAL DW\SCI\REPOSITORIO\FORMCONS FORM PADRÃO PARA TELAS DE CONSULTAS (GRID) DW\SCI\REPOSITORIO\FORMCADPASTA IGUAL AO FORMCAD, PORÉM CONTÉM PASTAS DW\SCI\REPOSITORIO\FORMDIGPASTA IGUAL AO FORMDIG, PORÉM CONTÉM PASTAS DW\SCI\REPOSITORIO\FORMSIMPLES FORM QUE CONTÉM O MÍNIMO DOS RECURSOS PADRÕES DA SCI DW\SCI\REPOSITORIO\FORMPROCESSO FORM UTILIZADO PARA ACOMPANHAMENTO DE PROCESSOS DW\SCI\REPOSITORIO\FORMACESSO FORM UTILIZADO PARA CONFIGURAR O VISUAL ACESSO

1.2.13) COMPONENTES PADRÕES:

São classes de objetos visuais (componentes) programados pela SCI, para serem utilizados em todos os nossos sistemas. Portanto, ao construir janelas, jamais devem ser utilizados os componentes da paleta Standard. Utilize os componentes Edit, Combo, Check, etc. da paleta SCI. Eles já estão programados para utilizar a nossa fonte de letras padrão, teclas de função padrões, etc. Também se for necessário criar uma classe de objetos nova, do tipo Edit, você deve criá-la sendo filha de uma classe Edit da SCI, e não filha de uma classe Edit direta do Delphi. Se você não estiver muito adaptado ainda com os padrões da SCI, é importante você estudar o código-fonte dos componentes padrões da SCI.

1.2.14) FORMULÁRIOS PADRÕES:

São janelas já desenvolvidas com os recursos padrões da SCI, que serão classes-pai das janela que você irá desenvolver em seu sistema. Ex: Se você precisar desenvolver um formulário para cadastro de empresas, você irá utilizar o uFormCad.Pas ou uFormCadPasta.Pas, que são formulários ideais para cadastros. Já se você precisar desenvolver uma tela para lançar valores em alguma das empresas cadastradas no uFormCad ou uFormCadPasta, você utilizará o uFormDig ou uFormDigPasta, que são formulários onde você primeiro seleciona o arquivo que você irá lançar, e depois você lança dentro desse arquivo. Todos os recursos padrões da SCI para cada tipo de janela já funcionarão automaticamente em seu formulário, sem necessidade de muita programação, bastando preencher alguns parâmetros. Eis as vantagens disso : 1. Maior rapidez na montagem do formulário. 2. Segurança : os recursos padrões já estarão programados e funcionando, sem necessidade de programar repetidamente. 3. Todas as telas do sistema terão o mesmo formato, economizando tempo de treinamento ao usuário final, que se acostumará mais rápido com as telas dos sistemas. Se você não estiver muito adaptado ainda com os padrões da SCI, é importante você estudar o código-fonte dos componentes padrões da SCI.

1.2.15) COMO UTILIZAR OS FORMS PADRÕES uFormCad e uFormDig:

Adicione uFormCad e/ou uFormDig ao Projeto. Quando quiser criar uma janela que seja estilo estas, vá até a opção File|New, na pasta do projeto e selecione frDados ou frSelecao.

uFormCad é uma janela ideal para digitação de dados cadastrais, tipo Cadastro Empresas, Usuários, etc.

uFormDig é ideal para telas de digitação de dados que necessitam selecionar uma tabela para digitação, tipo cadastro de plano de contas, funcionários, tela de lançamentos da Contabilidade, etc.

Após selecionar uma delas, você só precisará criar os edits para receber os dados, e informar alguns dados para que o form trate de maneira correta a tela:

Os Dados a serem informados para o form devem ser informados no evento OnCreate do Form. São Eles:

UFORMCAD:

wTabelaForm2: Você deve informar aqui qual é a tabela (Componente TTable) que contém os dados a serem digitados.

wChavePesquisa2: Aqui você deve informar qual é a chave principal de pesquisa da tabela (Geralmente é o código) ex: 'bdCodigo'

wChaveAlfabética2: Aqui você informa qual é a chave para ordenação alfabética. Ex : 'bdNome'

wComecaIncluindo: Aqui você informa se ao entrar na tela, já deve começar com uma inclusão. True ou False.

wEditChave2: Esta variável deve conter o componente Edit onde será digitada a chave para pesquisa. Se for informado, deve ser informado no evento OnShow, após o inherited. Se você não informar nada nessa variável, o sistema irá considerar o primeiro componente Edit colocado na tela como o Edit que contém a chave.

wUltimoEdit: Esta variável deve conter o componente Edit que ao passar por ele ocorrerá a gravação do registro. Se não for informado, o sistema irá 'pegar' o último Edit válido da tela.

wCarregaEdits: Aqui você informa uma procedure que irá carregar os dados de wTabelaForm2 para os edits, incluindo-se a chave. Deve ser informada obrigatoriamente. Deve ser uma procedure contendo um parâmetro do tipo Tform. Este parâmetro contém o objeto Form da tela. A procedure informada não deve pertencer ao Form.

wPesquisaChave2: Aqui deve ser informada uma Função que pesquise o banco de dados afim de saber se a chave digitada existe ou não. Se não for informada, o sistema irá pesquisar o conteúdo de wEditChave2. Esta função deve retornar Boolean, já informando se o registro existe ou não. É passado para ela um parâmetro contendo o objeto Form que está sendo processado. Esta função, assim como wCarregaEdits e qualquer outro procedimento ou função a ser criado como auxílio ao UFORMCAD, não devem pertencer ao Form; sua ligação ao form é o parâmetro do tipo Tform que é passado.

wLimpezaDados: Procedimento que é chamado quando o sistema precisa limpar o conteúdo dos Edits. Deve Ter as mesmas características dos outros procedimentos, tipo wCarregaEdits.

wNovaChave: Aqui você informa uma função que irá retornar uma nova chave quando o sistema precisar incluir um novo registro. Esta função deve retornar Variant. Se não for informada, o sistema irá pegar o conteúdo do wEditChave2 e somar 1.

wPodeIncluir: Informe aqui uma função que retornará boolean. Esta função deve analisar se os dados que estão na tela no momento poderão ser inclusos ou não.

wPodeExcluir: Semelhante ao wPodeIncluir, porém analisa a exclusão.

wGravaCampos: É o contrário de wCarregaEdits. Este procedimento deve gravar os dados dos edits em wTabelaForm2, incluindo-se a Chave.

wCompletaInclusao: Informe aqui um procedimento que é chamado após a inclusão de um registro. Serve para completar a inclusão, como por exemplo, criar arquivos após a inclusão de registros.

wCompletaExclusão: Semelhante ao wCompletaInclusão, porém é executado após a exclusão dos registros.

wChamaConsulta2: Procedimento que é chamado ao clicar no botão de consulta.

UFORMDIG:

Contém todos os procedimentos e funções acima mais:

wTabelaForm1: Semelhante ao wTabelaForm2, deve conter o componente TTable da parte de cima da janela UFORMDIG.

wChavePesquisa1: Deve conter a chave principal da wTabelaForm1. Ex: 'bdCodigo'

wChaveAlfabética1: Deve conter a chave que faz a ordem ficar alfabética. Ex: 'bdCliente'

wEditChave1: Deve conter o componente TEdit onde irá ser digitada a chave da primeira parte do UFORMDIG.

wCarregaSelecao: Procedimento que irá completar o momento em que é carregado um registro da wTabelaForm1.

wPesquisaChave1: Semelhante ao wPesquisaChave2, porém vale para a primeira parte de UFORMDIG.

1.2.16) UTILIZAÇÃO DE ARQUIVOS DE RECURSOS:

Os arquivos de recursos visam diminuir o tamanho do executável dos sistemas, sendo que cada vídeo ou imagem colocada em seu sistema aumentará consideravelmente o tamanho do executável, principalmente no caso da utilização do mesmo vídeo ou da mesma imagem em várias partes do sistema. Visando a organização e padronização de vídeos e figuras a Santa Catarina Informática adotou o seguinte padrão para seus arquivos de recursos.

1.2.16.1) - CRIAÇÃO DO ARQUIVO DE RECURSO

Cada sistema terá o seu arquivo de recurso, que levará o nome do executável mais a extensão ".RC", veja abaixo um exemplo do arquivo de recursos.

iSALVAR BITMAP "C:\DW\SISTEMA\RECURSOS\IMAGENS\iSALVAR.BMP" iEXCLUIR BITMAP "C:\DW\SISTEMA\RECURSOS\IMAGENS\iEXCLUIR.BMP" iFUNDO JPG "C:\DW\SISTEMA\RECURSOS\IMAGENS\iFUNDO.JPG" vSCI AVI "C:\DW\SISTEMA\RECURSOS\IMAGENS\vSCI.AVI"

Onde o primeiro parâmetro é o apelido do recurso que será utilizado no sistema, o segundo parâmetro é o tipo de imagem ou vídeo e o terceiro parâmetro é o caminho onde se encontra o recurso.

Este arquivo de recursos deverá ser compilado utilizando o arquivo BRCC32.EXE que está localizado no diretório "BIN" do delphi, após compilado será gerado um novo arquivo com o mesmo nome do arquivo ".RC", mas com extensão ".RES".

OBS: Para realizar esta compilação sem acessar o Prompt do MS-DOS, crie um arquivo de lote chamado COMPILA.BAT que receberá a seguinte linha de comando: C:\ARQUIVOS DE PROGRAMAS\BORLAND\DELPHI5\BIN\BRCC32 "Nome do executável do Sistema".RC

Diretórios padrões para Recursos

C:\DW\SISTEMA\RECURSOS - Diretório de Recursos do sistema C:\DW\SISTEMA\RECURSOS\IMAGEMS - Conterá os recursos em forma de imagens BMP, JPG etc. C:\DW\SISTEMA\RECURSOS\AVI - Conterá os recursos de vídeos no formato AVI

1.2.16.2) CRIAÇÃO DA DLL DE RECURSOS

O sistema acessará o arquivo de recursos através do arquivo RECURSOS.DLL localizado no diretório C:\DW\SISTEMA\DLL, esta DLL conterá o arquivo de recursos já compilado, para criar esta DLL, vá ao menu File/New/DLL, veja abaixo o exemplo:

Library Recursos;

{$R C:\DW\Sistema\Recursos\"Nome do Sistema".RES}

Begin

End.

Após compilado o projeto da DLL será criado o arquivo RECURSOS.DLL.

Para acessar a DLL no sistema deverá ser utilizado a função LoadLibrary passando como parâmetro o caminho do arquivo RECURSOS.DLL, esta função retornará o Handle do arquivo que deverá ser armazenado em uma váriavel, para ser utilizada nos demais processos do sistema como por exemplo ao carregar uma imagem em um TSppedButton:

spSalvar.Glyph.LoadFromResourceName(Handle,Apelido do Recurso);

após utilizado os recursos você deverá descarregar a DLL da memória utilizando o procedimento FreeLibrary passando como parâmetro o Handle armazenado na váriavel.

LoadLibrary e FreeLibrary são utilizadas uma única vez na execução do sistema, a primeira no começo e a segunda no final da execução.

Utilitário - Lista dos paths para utilização na instalação de componentes:

$(DELPHI)\Lib;$(DELPHI)\Bin;$(DELPHI)\Imports;$(DELPHI)\Projects\Bpl;C:\DW\SCI\COMPONENTES;C:\DW\SCI\REPOSITORIO\FORMCAD;C:\DW\SCI\REPOSI C:\DW\SCI\COMPONENTES\DBEXPRESSSCI; c:\dw\sucessor;c:\dw\sci\repositorio\calendario;c:\dw\sci\repositorio\calculadora

--Fábio Schulter 12h47min de 31 de julho de 2009 (UTC)