

Este é o nosso primeiro trabalho para 2008, espero que este ano seja marcado por uma maior participação comunitária à nossa tentativa de expor mais trabalhos técnicos de todos os níveis. Vamos nos antecipar à chamada para apresentação de trabalhos ao nosso evento anual que nesse ano será em Juiz de Fora-MG, a riqueza de tópicos é muito extensa, colabore você também, escolha uma e venha participar conosco. Estou fazendo muita força e me sinto gritando sozinho para manter nossa comunidade ativa, mas, para continuar espero o engajamento de diversos outros colegas.

Introdução

As partes que compõe

Botão OK

Botão Cancelar

A WINDOW

Método PrimeUpdate

Método TakeCompleted

Validação de Campos

Chamando um FORM sem usar um Browse

Criando um procedimento SOURCE

Editando o FORM

Introdução

Muito provavelmente a expectativa agora seria a de olhar para um Browse, mas achamos mais oportuno antecipar o template FORM por ser bem mais simples do que o Template Browse e com isto nos remeter a uma melhor compreensão do todo.

Em primeiro lugar, o que é um FORM? Claro que se você tem utilizado o CLARION sabe que o Form é utilizado para realizar a manutenção dos registros de uma tabela (incluir, alterar, excluir ou visualizar) da base de dados. Considere que a WINDOW não está conectada na sua base de dados como um todo, considere também, que o Browse tem uma visão da base de dados, então o Form é o lugar onde os registros podem ser editados.

As partes que compõe o Form

O Form atualmente é feito em quatro partes:

- 1) Window, baseada na classe WindowManager;
- 2) O botão OK (também chamado de SAVE BUTTON);
- 3) O Botão Cancelar;
- 4) Validação do Registro

Existem vários jeitos de explanar cada uma das quatro partes acima enumeradas, o fato é que elas trabalham em conjunto para chegar ao objetivo proposto. Como a Window contém muito da funcionalidade e oferece mais pontos embutidos para incluir códigos, poderia iniciar por ela. Por outro lado, iniciarei com uma parte mais simples.

O Botão OK

Quando o botão OK é pressionado então o registro que estava sendo editado começa a ser gravado no disco. Como você pode imaginar o comportamento é muito genérico, por isso, apenas o código é armazenado na biblioteca ABC. Tudo que a Window tem de saber é qual o controle na tela é de fato o botão OK. Isto é feito armazenando um EQUATE para o botão OK em uma das propriedades da WINDOW. O código é gerado dentro do método INIT e fica mais ou menos assim:

Código:

```
SELF.OKControl = ?OK
```

Lembre-se de que SELF é apenas um jeito para o compilador saber exatamente de quem estamos nos referindo. Neste caso como está dentro do Método "ThisWindow.Init" então Self se refere a ThisWindow.

OkControl é uma propriedade do objeto ThisWindow. Colocando de uma outra forma, o objeto ThisWindow tem um conjunto de propriedades e OkControl é uma delas.

Agora que a Window sabe qual controle é o botão OK, ela automaticamente responderá conforme esperado quando o botão OK for pressionado.

O Botão Cancelar

De forma similar ao botão OK, temos de dizer para a Window qual o controle que representa o cancelamento da operação. O código para fazer isto também está incluído no método INIT do objeto ThisWindow, declarado mais ou menos assim:

Código:

```
SELF.AddItem (?Cancel, RequestCancelled)
```

Desta vez a Window utiliza um método para registrar o botão cancelar e marca a ação para o botão.

No caso do botão OK, por ser uma propriedade da Window, não é necessário marcar a ação para o botão.

Como o botão cancelar é tratado como um item na window pode existir múltiplos botões cancelar registrados. Esta não é uma questão lingüística ou algo do gênero, é simplesmente a forma que a classe WINDOWMANAGER foi concebida.

A WINDOW

Muito bem, agora a Window já sabe qual é o botão OK e o botão Cancelar. Como isto pode ajudar? Todo o código é colocado dentro da biblioteca, que é uma coisa muito boa, uma vez que é genérico para todos os FORMS.

Um FORM é apenas uma WINDOW fantasiada. Isto quer dizer que temos três pontos embutidos em comum onde podemos colocar códigos. Lembra-se quais? INIT, TakeEvent (ASK) e Kill. Existem também dois adicionais, específicos do FORM, que podem ser de muita utilidade.

Método PrimeUpdate

O primeiro método específico do FORM que a Window chama é o PRIME UPDATE. Este método é chamado (pelo template) de dentro do método ThisWindow.Init.

Em uma instalação multi-usuária, (todo programa CLARION assume que está sendo executado em uma instalação multi-usuária), quando entra em um FORM, o estado corrente do registro é "SALVADO". Quando o botão OK for pressionado o registro original é re-lido do disco e comparado com a versão original. Se o registro original não estiver idêntico ao lido será gerado um erro "RECORD CHANGED BY ANOTHER STATION". Assim se duas pessoas estiverem editando o mesmo registro no mesmo tempo, então a segunda pessoa não poderá inadvertidamente sobre-escrever as alterações realizadas pela outra pessoa.

Este salvamento do registro original é realizado pelo método PrimeUpdate. Um bom local para realizar cálculos nos campos dos registros. (depois de chamar o PARENT)

Se você quiser fazer os cálculos quando o registro for incluído, então utilize o método PrimeFields. PrimeFields é chamado pelo PrimeUpdate, mas somente se um registro for incluído.

Citação:

Ponto Embutido – Para setar valores nos campos em todos os casos, utilize o método PrimeUpdate, depois da chamada PARENT.

Exemplo (dentro do método PrimeUpdate). Neste exemplo se você quiser fazer alguma coisa somente se for uma inclusão então você pode testar a propriedade Self.Request.

Código:

```
ReturnValue = Parent.PrimeUpdate  
Fil:LastEditTime = Clock()  
Fil:LastEditDate = Today()  
If Self.Request = InsertRecord then  
    Fil:CreatedBy = ds_CurrentLogin(Appnum)  
End
```

Citação:

Ponto Embutido – Para colocar valores nos campos somente quando for uma inclusão, utilize o método PrimeFields.

Código:

```
Fil:CreatedBy = ds_CurrentLogin(Appnum)
```

Porque o código do método PrimeFields somente será executado se a propriedade REQUEST for uma inclusão (igual a InsertRecord).

Agora uma questão óbvia. Se você pode testar InsertRecord no método PrimeUpdate, por que iria embutir códigos para o método PrimeFields? Bem, um pouco desconcertante, de fato, pode-se não utilizar o método PrimeFields, contudo devemos ser sempre consistentes e lembrar que ao elaborar qualquer aplicação esta terá o resto de sua existência dedicada exclusivamente a manutenção, por parte do próprio programador ou de futuros programadores e, quanto mais consistente for o código, mais fácil tanto de encontra-lo quanto de intervir sobre o mesmo.

Citação:

DICA I – Se você receber a mensagem de erro “Changed by another station” e tiver a certeza de que não houve alteração por outra estação deste registro, então é provável que você tenha algum código embutido antes que PRIMEUPDATE tenha sido chamado. Em outras palavras, o “REGISTRO ORIGINAL” não foi guardado com os valores corretos.

Citação:

DICA II – Porque PRIMEUPDATE é chamado de dentro do método INIT, qualquer coisa que seja inserido em PRIMEUPDATE poderia ser colocada em INIT ao invés de PRIMEUPDATE. Seja cuidadoso! Se for colocado em um ponto que possa ser movido mais tarde para depois da chamada PRIMEUPDATE ou até antes desta chamada. Portanto, em longo prazo devemos utilizar os locais mais lógicos, utilizando o método PRIMEUPDATE e o método PRIMEFIELDS.

Apagando Registros

Uma das vantagens dos templates é que eles permitem que sejam feitas a mesma coisa de diferentes formas. Um exemplo disto é quando vamos excluir um registro. Quando o usuário clicar no botão EXCLUIR em um BROWSE então uma de três possíveis métodos de exclusão pode ser chamado.

1) – O FORM é aberto e todos os campos são desabilitados. Se o usuário clicar no botão OK o

registro é apagado. Já, se clicar no botão cancelar, o registro NÃO será apagado. Esta forma nos leva aos dias do CLARION FOR DOS e parece estar fora de moda no momento. Acho que pode ficar meio confuso ao usuário final, talvez por isso não seja tão usado hoje.

2) – Uma caixa de mensagem aparece para que o usuário responda se realmente quer apagar o registro selecionado. Este é o comportamento padrão utilizado pelo template.

3) Nenhuma mensagem aparece, o registro é diretamente apagado.

O importante é salientar que independente de qual método for empregado é que quem efetivamente apaga o registro é o FORM. Nos dois últimos casos o FORM ficou invisível mas foi ele quem efetivou a ação.

Com o primeiro método o DELETE é efetuado pelo método TAKECOMPLETED. (Este método será descrito em seguida) Por outro lado, com o segundo e com o terceiro método, o trabalho é realizado com o método PRIMEUPDATE e, como já sabemos, este método é chamado de dentro do método INIT.

Portanto, se existe alguma coisa a ser feita antes do registro ser apagado, então provavelmente você elegeria embutir código no método INIT antes de chamar o método PRIMEUPDATE. Contudo o melhor lugar para isto é dentro do método PRIMEUPDATE antes de chamar o PARENT, isto porque não existem pontos embutidos entre a abertura dos arquivos e a chamada para o método PRIMEUPDATE. É preciso colocar algum código para testar se o registro atual será apagado Por exemplo:

Código:

```
If Self.Request = DeleteRecord then
    ! Faça alguma coisa aqui
    ! Para abortar o delete, retorne com LEVEL:NOTIFY
End
```

Método TakeCompleted

Quando o botão OK for pressionado, então o método que faz o trabalho chama-se TAKECOMPLETED. Se você quiser colocar algum código antes ou depois que o registro for gravado, o melhor local é aqui.

Um exemplo antes que o registro tenha sido escrito poderia ser:

Código:

```
Fil:SaveDate = Today()
Fil:SaveTime = Clock()
ReturnValue = Parent.TakeCompleted()

Ou

If UserLevel <> Supervisor then
    Cycle
End
ReturnValue = Parent.TakeCompleted()
```

No primeiro exemplo, estamos simplesmente completando o registro antes de salva-lo. No segundo fazemos um teste para saber se o usuário pode completar a operação.

Citação:

DICA: É tentador colocar o código de validação no próprio botão OK, porém se o usuário não gravar e pressionar o botão cancelar, recebe uma mensagem “Você não gravou o registro, deseja fazê-lo agora?” e se ele responder que sim neste momento, nada do que estiver embutido no botão OK será executado. Portanto prefira usar sempre o método TAKECOMPLETED para este tipo de código.

Validação de Campos

As regras para a validação dos campos de arquivos devem ser colocadas no dicionário de dados. O template gera o código de validação dentro dos GLOBALS da sua aplicação. Você pode adicionar seu código de validação específico se quiser. Para tanto, você deve ir ao ponto global da aplicação, denominado FIELDLEVELVALIDATION, selecionar a tabela em questão e o campo para colocar o seu código.

Contudo se você fizer isto, terá ainda de gerar os seus próprios códigos de validação do formulário. Francamente. Não gaste o seu tempo. Faça a validação diretamente no dicionário sempre que possível e não terá de colocar código algum para o método TAKECOMPLETED.

O template gera código, para ativar as validações, dentro do método TAKEACCEPTED, na WINDOW. Isto nos permite desligar as validações para um formulário específico quando necessário.

Chamar um FORM sem usar um BROWSE

Muitas vezes são postadas perguntas nos Fóruns ou Newsgroups sobre como adicionar ou alterar um registro sem usar um BROWSE. Usualmente isto ocorre com arquivos de um único registro. Normalmente você quer ir direto do FRAME para o FORM.

Esta questão parece estar relacionado ao FORM, então colocar este texto aqui pareceu ser o melhor lugar. Contudo, a resposta utiliza o código do objeto FILEMANAGER. (O qual nós veremos em um artigo futuro para esta coleção). Então o leitor pode querer ver esta questão mais tarde, aguardando o referido artigo.

Existem dois preparos que podem ser feitos. Pode adicionar o código diretamente no FORM ou ainda fazer um pequeno procedimento SOURCE o qual essencialmente duplica os efeitos de um BROWSE. As duas formas são boas e tem vantagens e desvantagens.

Citação:

DICA: Você pode setar a variável GlobalRequest no Frame e então chamar uma nova Thread. Isto porque esta variável é THREARED e uma nova cópia é criada a cada nova THREAD.

Criando um Procedimento SOURCE

A primeira possibilidade seria criar um pequeno procedimento utilizando o TEMPLATE SOURCE, que basicamente duplica os efeitos de um BROWSE. Para efeitos do presente exemplo, assume-se que esta querendo editar um arquivo de um único registro. Se o arquivo não contém o registro, este deve ser criado. O código do procedimento SOURCE. O código ficaria mais ou menos assim:

Código:

```
Code
If Access:Control.Open() = Level:Benign then
  Access:Control.UseFile()
  If Records(Control)=0 then
    GlobalRequest = InsertRecord
    If Access:Control.PrimeRecord() = Level:Benign then !Inicializa um
registro
      UpdateControl
    end
  Else
    GlobalRequest = ChangeRecord
    Set(Control)
    Access:Controle.Next()
    UpdateControl
  end
End
```

Se quiser apenas adicionar registros, então fica um pouco mais simples:

Código:

```
Code
If Access:Control.Open() = Level:Benign then
  Access:Control.UseFile()
  GlobalRequest = InsertRecord
  If Access:Control.PrimeRecord() = Level:Benign then !Inicializa um
registro
    UpdateControl
  End
  Access:Control.Close()
End
```

O método é chamado para verificar como devem ser inicializados os campos do registro do arquivo e também toma cuidado com algum campo de auto-incremento.

Uma vantagem desta técnica é que o FORM não é alterado de jeito algum. Então é possível utilizar o procedimento FORM em conjunção ao BROWSE. A desvantagem é que o código é fisicamente separado do FORM. Se o FORM nunca for chamado de um Browse então é mais fácil colocar este código diretamente no FORM.

Editando direto do FORM

Basicamente o código escrito não é alterado. Mas nós precisaremos atentar para qual ponto embutido colocar. Também necessitaremos criar uma variável local para abrir e fechar o arquivo.

Primeiro vamos criar uma variável de nome OUIOPEN do tipo BYTE.

Segundo, adicionamos o código abaixo no método INIT, diretamente depois da linha:

Código:

```
GlobalErrors.SetProcedureName('UpdateControl')
```

E antes da linha:

Código:

```
Self.Request = GlobalRequest
```

Para finalizar o código atente que o código em negrito é gerado diretamente pelo template:

Código:

```
GlobalErrors.SetProcedureName('UpdateControl')
If AccessControl.Open() <> Level:Benign then
  Return Level:Notify
End
OurOpen = 1
Access:Control.UserFile()
If Records(Control) = 0 then
  GlobalRequest = InsertRecord
  Access:Control.PrimeRecord()
Else
  GlobalRequest = ChangeRecord
  Set(Control)
  Access:Control.Next()
end
Self.Request = GlobalRequest
```

Em terceiro lugar, precisamos fechar o arquivo. Então no método KILL adicione o seguinte código:

Código:

```
If OurOpen = 1 then
    Access:Control.Close()
End
```

Caso queira uma rotina que apenas adicione novos registros, então coloque o seguinte código:

Código:

```
GlobalErrors.SetProcedureName('UpdateControl')
If AccessControl.Open() <> Level:Benign then
    Return Level:Notify
End
OurOpen = 1
Access:Control.UserFile()
GlobalRequest = InsertRecord
Access:Control.PrimeRecord()
Self.Request = GlobalRequest
```