

O Edit-In-Place é uma idéia para você entrar ou alterar informações diretamente em um Browse, sem ter de abrir um formulário para o respectivo registro. Em outras palavras, o Browse deve se comportar mais como uma planilha eletrônica permitindo que os dados que sejam apresentados e que possam ser alterados.

O Edit-In-Place por um longo tempo não foi incluído no CLARION, muito embora a linguagem o pudesse suportar desde o início, os templates não traziam tal implementação, até que os templates ABC foram lançados com a versão 4 do Clarion. Enquanto a sua inclusão foi muito aplaudida a execução não recebeu tantos elogios assim. A maleabilidade na entrada de informações era muito restrita e os recursos para intervir também deixavam muito a desejar. Com a versão 5 do Clarion é que foi removido da classe Browse e criada uma classe específica e com o Clarion 5.5 foram ampliados os recursos.

Neste trabalho nos concentraremos nas implementações correntes. Usuários do Clarion 5 serão capazes de utilizar mais informações, embora algumas das novas partes estejam faltando. Já os usuários do Clarion 4, não poderão utilizar tais informações.

Vantagens

Talvez o problema mais significativo do EIP é que todos os comparava com uma planilha eletrônica. Mas as aplicações com base de dados em geral não são carregadas completamente como uma planilha. Em outras palavras, em uma planilha todas as linhas são carregadas para a memória de uma só vez, enquanto que em uma aplicação cada registro é carregado individualmente e poucas são as chances de tratar como um conjunto de informações. Isto quer dizer que em uma planilha existe um botão para gravar. Enquanto que em uma aplicação com base em dados a expectativa é que cada registro seja salvo individualmente. Mas uma planilha tem a característica de “desfazer” algumas ações que se pensar em linhas individuais seria impossível.

Então não existe uma forma definida de responder como o EIP faria isto. Muitas questões ficam sem respostas pelos desenvolvedores. Vejamos:

Quando um registro deve ser salvo? No final de cada linha? Depois de cada coluna? Quando alterar um registro na menor porção de informações possível, uma coluna é provavelmente o melhor momento para salvá-lo. Agora se o registro tiver cinco campos, isto será muito pouco eficiente, imagine gravar o registro a cada entrada de um dos seus campos. E se o usuário navegar em vários sentidos dentro da planilha, para baixo, para cima, para trás ou para frente que seria ou se esperava sempre? Algumas vezes o usuário sobe no “Browse” para alterar informações em registros que já foram alterados.

Qual o melhor momento para verificar a consistência das informações? Depois de todas as alterações? A cada linha? Depois de cada coluna? Que ao se alterar uma coluna afeta o conteúdo de outras, (como por exemplo, mostrar o resultado), isto facilmente resolveríamos colocando campos calculados como funcionam as fórmulas nas planilhas, mas o pensamento do desenvolvedor é em uma seqüência de formulário.

Quanto às inclusões de registros, você vai tratar como se fossem as alterações? Então incluiremos uma linha na posição do cursor ou no final da planilha? Qual momento para salvar isto tudo? Vamos supor que será ao final de cada linha, veja

que tem mais de uma forma de ir até lá. Então se o usuário incluiu um novo cliente e os últimos 40 campos ainda estivessem em branco. Como o usuário completaria as informações daquela linha? Se o “Browse” tem somente uma porção dos campos, como fazer uma inclusão completa?

Como ativar o modo de edição? Com a rápida digitação do usuário? Quando ele pressionar a tecla ENTER? Quando ele der um “duplo-clique” no mouse? Tipicamente uma planilha aceita as três formas.

Iniciaria a edição na coluna corrente ou só no início da linha?

Outro problema é que no “Browse” tem somente uma parte das informações do registro e em determinada situação do Edit-In-Place será necessário mostrar todas as informações daquele registro para entrar em modo de edição ou de inclusão.

Um formulário, sendo uma janela, tem acesso em todos os controles. Alguns controles podem ser convertidos para o EIP outros não têm como. Por exemplo, Entry, Drop-Downs e check box, podem ser mostrados em um EIP de forma elegante, mas como fazer para Radio Buttons, Lookup Buttons ou Calendários?

Como os “Browses” que mostram informações de outros arquivos que não do arquivo primário fazem? Por exemplo, em um browse de pedidos onde encontramos a coluna de nome do cliente, então você permite que o usuário altere aqui o nome do cliente? Como fazer o usuário entender de que aqui não pode editar o nome do cliente?

Desta lista de complicadas perguntas, para as quais existem poucas respostas simples, temos de encontrar soluções genéricas. Conseqüentemente o template ABC disponibiliza algumas opções. Entretanto, veremos mais adiante como é possível estender as classes para responder algumas das questões.

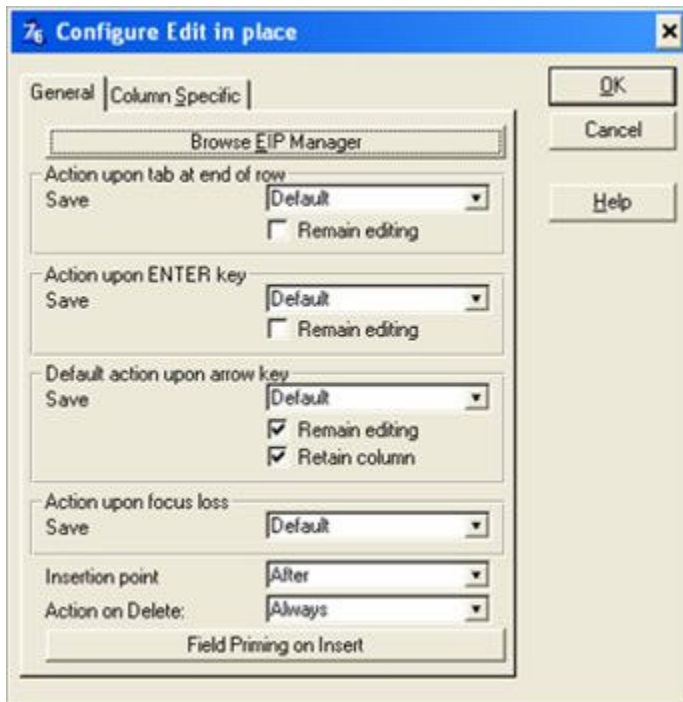
Citação:

Uma recomendação. Tente fazer as suas escolhas e implemente sua aplicação de uma só forma tanto quanto possível. Nada é mais confuso para um usuário final do que trabalhar de um jeito em um lugar e de outro jeito em outro. Unificando os procedimentos seu usuário tenderá a habituar-se à filosofia de trabalho de forma mais rápida.

Uma rápida olhada no template EIP.

A documentação que acompanha o CLARION sobre EIP é muito pouca, então uma olhada diretamente no template é um bom lugar para começar.

Em primeiro lugar, vamos dar um giro sobre o EIP. Na guia das propriedades dos botões de atualização de um browse “Update Browse” tem a opção para marcar a utilização do EIP e um botão “Configure o EIP”. É óbvio que para avisar nossa aplicação para utilizar o EIP devemos clicar nesse botão, contudo as telas seguintes já não são tão óbvias assim.



Browse EIP Manager - Administrador do EIP

Aqui você pode cancelar a classe EIP. Mais à frente deste trabalho discutiremos com mais detalhes esta classe. Se você fizer a sua própria classe então é aqui que deve ser dito para o template.

Citação:

Nota: Você pode também dizer para o template substituir todas as classes BrowseEIPManager de forma global. Clique nas propriedades globais da aplicação, então através da guia CLASSES, clique no botão Browser.

Action upon tab at end of row – Ação sobre a guia “para fim de linha”

Action upon ENTER Key – Ação sobre a tecla ENTER

Default Action upon arrow key – Ação padrão sobre as setas de navegação

Action upon focus loss – Ação sobre perder o foco

Veremos com mais detalhes cada uma destas ações mais adiante, por enquanto basta dizer que elas agem quando o registro será salvo e o que fazer depois de salva-lo.

Insertion Point

Quando o usuário pressiona a tecla INSERT, ou seleciona inserir através do menu popup ativando com o botão direito do mouse, então esta opção faz o balanceamento da ação. Isto diz para a aplicação se deve abrir uma linha no ponto corrente do browse (antes ou depois do registro iluminado) ou ainda se deve ir ao final da lista para iniciar a operação.

Action on Delete

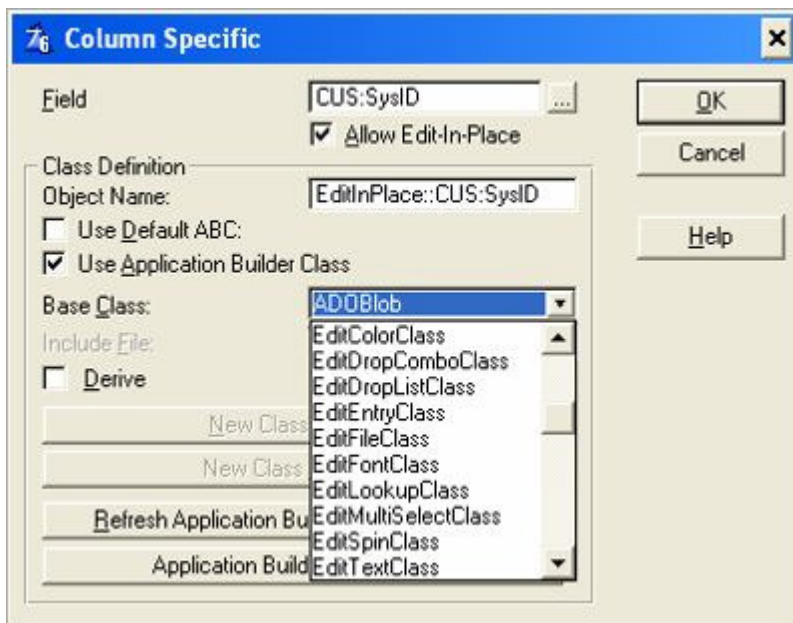
Determina o comportamento para a ação de excluir.

Column Specific – Coluna específica

Aqui se encontra a forma como o desenvolvedor pode realmente se comunicar com o EIP. No momento vamos evitar falar sobre os códigos e pontos embutidos que podemos intervir,

veremos isto um pouco mais adiante, por enquanto vamos analisar as opções disponíveis.

Na seção de coluna específica definimos o comportamento de cada um dos campos do Browse.



O mais simples de tudo é desligar o EIP em cada coluna, através da check box "Allow Edit-In-Place" como pode ser visualizado na imagem acima, ou ainda, marcar nossa escolha para utilizar na entrada do respectivo campo. Para tanto, desligue "Use Default ABC" pois a classe padrão é Entry e selecione uma das classes previamente elaboradas. Digamos que para o campo "CUS:SysID" gostaríamos de utilizar a classe EditDropListClass, que trata-se de um controle DROPLIST dentro do EIP. De tal forma que quando o usuário entrar editando neste campo verá um controle DROPLIST automaticamente.

Vamos olha cada uma das opções com um pouco mais de detalhes.

EDITCHECKCLASS

Esta classe mostra ao usuário uma caixa de marcar, isto é muito utilizado para campos onde a variação de valor é do tipo Verdadeiro ou Falso ou ainda 1 ou 0. Infelizmente existe um problema no template que nos obriga a intervir manualmente.

Como essa classe é derivada da EditClass que está perfeita em seu funcionamento, A EditClass marca a propriedade prop:text (no método init). Entretanto para a classe CheckBox esta propriedade deve ser cancelada. Devemos elaborar então um código próprio para alcançarmos este objetivo. Use o editor de códigos embutidos para encontrar o código como pode ser visto abaixo. (PRO:ATIVO é o nome do campo)

Código:

```
EditInPlace::PRO:ATIVO.Init PROCEDURE(UNSIGNED Fieldnumber, UNSIGNED
Listbox, *? UseVar)
Code
PARENT.Init(FieldNumber, Listbox, UseVar)
SELF.FEQ{Prop:Text} = '' !Adicione esta linha de código
```

(Adicione a linha que inicia com "SELF.FEQ..." acima, logo abaixo do PARENT CALL)

Outro ponto do checkbox está no fato de que você gostaria que seu usuário visualizasse o formato de uma checkbox e não o conteúdo do campo em si. Por exemplo: Digamos que o

campo PRO:ATIVO variasse o conteúdo em 0 para Falso e 1 para verdadeiro, então para uma maior clareza, o usuário irá ver 1 ou zero, enquanto que o ideal seria que ele visualizasse uma caixa checada ou não checada. Também podemos resolver isso rapidamente.

Abra o formatador do LISTBOX e localize a coluna relativa ao PRO:ATIVO, passe a picture para **@p p** esta fórmula fará com que nada seja mostrado em relação ao conteúdo do campo. Agora na guia "Appearance" no grupo Icon deixe marcada a opção "Transparent". Feche o formatador e abra então as propriedades do LISTBOX em sua guia ACTION. Pressione em "Browse Box Behavior" e vá para a guia ICON. Ali aparecerá uma lista dos campos que tem marcado para utilizarem uma imagem agregada ao valor. Entre em suas propriedades e em Default coloque "uncheck.ico" agora inclua uma linha condicional como na imagem abaixo:



Tenho certeza de que você ficará muito satisfeito com o resultado.

EditColorClass

Esta classe permite ao usuário optar por uma cor através de um botão de LOOKUP. Se uma cor for marcada haverá o retorno de uma de duas possíveis formas:

- a) – se a cor for padrão como color:red ou color:blue, então o retorno será assim;
- b) – se a cor não for padrão, então o retorno será no formato R : x, G : y, B : z, onde x, y e z são números no intervalo de 0 até 255.

Você notará que a classe não tem validação alguma, mais tarde veremos uma forma de validar a entrada e até mesmo tornar o LOOKUP mais amigável.

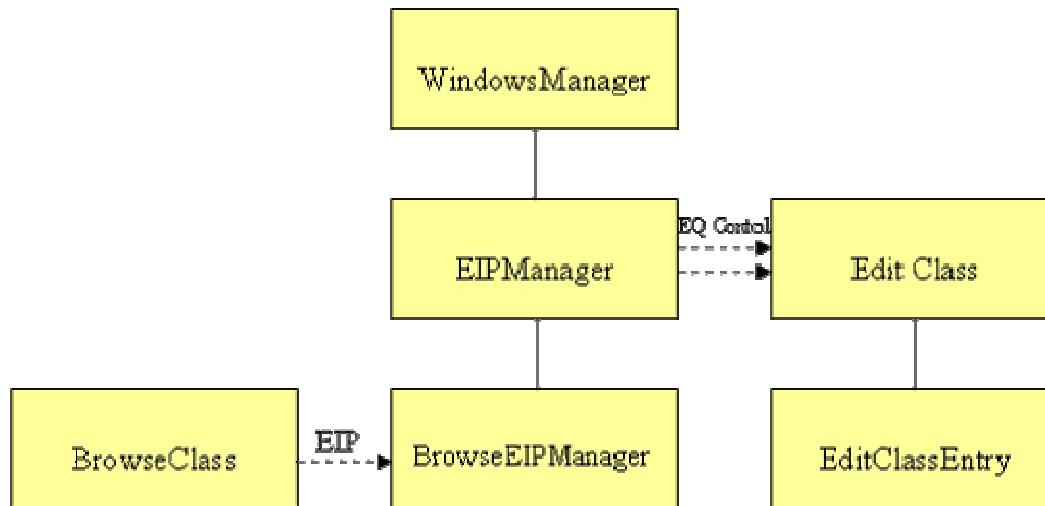
Outras classes incluídas:

- EditDropComboClass
- EditDropListClass
- EditEntryClass
- EditFileClass
- EditFontClass
- EditMultiSelectClass
- EditSpinClass
- EditTextClass

Um esboço do ABC EIP

Chegou o momento de olharmos para as classes EIP dentro do ABC. Quando você olha pela primeira vez o conjunto de recursos EIP e adiciona ao ABC a reação é imaginar que é uma loucura, é verdade, é muito complexo mesmo, por outro lado se pensarmos como interação de objetos, será muito mais simples ir compreendendo.

Vamos olhar o diagrama abaixo juntos.



Lendo o diagrama da esquerda para a direita, tudo começa com o BrowseClass. Este contém uma propriedade na qual ele mesmo chama o objeto EIP. EIP é baseado na classe BrowseEIPManager. A classe BrowseEIPManager é derivada da classe EIPManager. Em outras palavras BrowseEIPManager herda todas as propriedades e métodos da EIPManager. A classe EIPManager é derivada da classe WindowsManager.

Significa que a classe BrowseEIPManager possui uma Window. Ele tem uma window internamente completamente transparente sobre a Window que o contém. Isto é necessário porque todos os códigos de formulário (FORM) estão incluídos na classe WindowsManager. Lembre-se de que o EIP funciona como um pequeno FORM. Como pode haver muitos Browsers sobre uma Window, mas somente um FORM, torna-se necessário ter uma Window para cada Browse com EIP.

Para cada coluna no LISTBOX, a classe EIPManager contém um objeto baseado na classe EditClass. Porque existe um conjunto de colunas no Browse tem de haver uma Queue para armazenar todos os objetos. O nome desta queue é EQ. O campo dentro da queue que aponta para a classe EditClass denomina-se Control.

A última parte desta equação é a classe especialista EditClass. Lembre-se de que existe um para cada coluna do Browse e cada coluna pode ser baseada em um diferente controle. Assim existe uma diferente EditClass para cada controle. A mais básica delas é a EditEntryClass, a qual usa uma simples caixa de entrada de dados, mas existe também um conjunto de outras como EditTextClass, EditCheckClass, EditSpinClass entre outras.

A idéia é para que você possa dar para cada coluna um comportamento diferente, mas padronizado, dessa forma utilizando uma diferente EditClass para ela. Claro, elas são muito similares, mas uma Checkbox, por exemplo, comporta-se um pouco diferente de um drop-down ou mesmo de uma caixa de entrada (entry).

A dificuldade em descrever como o EIP trabalha é porque existem muitas partes, torna-se difícil saber por onde começar. Como já enunciado anteriormente “conhecendo o todo, podemos compreender os pedaços”, mas para estudar os pedaços, temos de levar algumas coisas como confiáveis. É recomendado estudar bem o trabalho até aqui por diversas vezes. Porque se ganha mais compreensão em cada vez que analisamos e assim cada vez mais passa a fazer sentido.

Vamos olhar cada um dos componentes e ver como eles vão trabalhando juntos para prover funcionalidades ao EIP.

BrowseClass

Se pensarmos bem sobre a classe BrowseClass, poderíamos dizer que é uma parte essencial do EIP. Já pensou fazer um EIP em uma listbox sem utilizar um Browse? Bem, talvez vejamos isto mais tarde, por enquanto vamos assumir que estamos falando especificamente

à respeito do Browse EIP.

As funcionalidades que um Browse provém são dobradas. Primeiro quando a Window é aberta alguns recursos necessários tem de ser iniciados. Então quando o usuário quiser provocar o EIP o browse é o responsável por executá-lo e processá-lo.

Iniciar

A classe BrowseClass contém um número de propriedades que afeta como o EIP se comporta. Importante notar que estas propriedades podem ser alteradas pelo programador e assim o comportamento visual pode ser alterado em tempo de execução. Assim é possível alterar o comportamento de um jeito ou de outro e nós veremos em um minuto como é.

Todas as propriedades e métodos seguintes são tipicamente marcados no método INIT do Browse. (BRW1.Init). Este é o local onde o template colocará as chamadas. Entretanto lembre-se de que pode alterar os valores em qualquer estágio. Obviamente, temos de tomar muito cuidado com essa atitude porque isto altera o comportamento da edição. Mas existem casos que temos de alterar as propriedades para obter um melhor funcionamento.

Citação:

ATENÇÃO – O Browse tem TRÊS métodos INIT. Um que é iniciado com a lista de parâmetros (SIGNED ListBox, *STRING Posit, VIEW V, QUEUE Q, RelationManager RM, WindowManager WM) outro com (IListControl IC, VIEW V, BrowseQueue LQ, RelationManager RM, WindowManager WM) e o último com (VIEW V, RelationManager RM, <SortOrder SO>).

Propriedade AskProcedure

Vamos supor que temos uma variável no browse chamada PodeEditar. Baseado no valor de PodeEditar o usuário é autorizado ou não a editar o registro selecionado. O correto “embed” para verificar este campo antes de tomar a decisão é no método da classe BrowseClass denominado ASK (BRW1.ASK). Isto porque este método é chamado um pouco antes de iniciar com a edição e o mais importante é que neste momento você saberá se o usuário está tentando incluir, alterar ou excluir o respectivo registro. Note ainda que o seu código deva estar ANTES da chamada PARENT.

Código:

```
If Request=InsertRecord or Request=DeleteRecord then
  Brw1.AskProcedure = 0 ! Permite editar
Else
  If Queue:Browse:1.PodeEditar = 1 then
    Brw1.AskProcedure = 0 !Pode editar
  Else
    Brw1.AskProcedure = 1 !Não pode editar
  end
End
```

É possível também conhecer a coluna corrente selecionada testando a propriedade **?Browse:1{prop:column}**.

Note que no exemplo, foi descrito como BRW1 sendo o nome do objeto BrowseClass, já **?Browse:1** é a referência relativa (equate) para o ListBox e ainda Queue:Browse:1 é o nome da queue do respectivo ListBox. Estes nomes são gerados pelo template e pode estar alterados de browse para browse.

Outro exemplo

Existem outras situações comuns, digamos que para incluir e excluir registros vamos usar um FORM, mas para alterar vamos utilizar o EIP.

O código para isto é muito similar, mas precisa usar um pequeno truque. O segredo aqui é preencher nas propriedades dos botões de chamada de atualização do Browse um

procedimento FORM e ainda marcar a utilização do EIP. A propriedade AskProcedure determina se o FORM ou o EIP será usada. E novamente coloque o seguinte código no método Ask da classe BrowseClass.

Código:

```
If Request=InsertRecord or Request=DeleteRecord then
  Brw1.AskProcedure = 1 !Vá para o FORM
Else
  Brw1.AskProcedure = 0 !Vá para o EIP
end
```

Um último exemplo

Agora vamos conhecer como selecionar entre usar o FORM ou o EIP e ainda condicionar a alteração conforme o conteúdo do campo PodeEditar, mas como fazer tudo isto junto?

Bem, no caso o que desejamos é abortar o procedimento caso para acomodar a terceira situação. Então, no mesmo embed, antes da chamada para o pai do objeto (parent call) vamos colocar o nosso código. Em outras palavras para a alteração, ao invés de ajustar a propriedade AskProcedure, simplesmente retorne a chamada deste procedimento (Ask).

Código:

```
If Request=InsertRecord or Request=DeleteRecord then
  Brw1.AskProcedure = 1 !Vá para o FORM
Else
  If Queue:Browse:1.PodeEditar = 1 then
    Brw1.AskProcedure = 0 !Vá para o EIP
  Else
    Return ReturnValue !Retorne da chamada deste método
  end
end
```

Conclusão:

O método ASK da Classe Browse é o lugar onde devem ser colocados todos os códigos para decidir se a edição é permitida ou reservada ou se utilizará EIP ou FORM.

ArrowAction – Propriedades

Esta propriedade cuida do que acontece se as setas para cima ou para baixo forem pressionadas enquanto o usuário estiver utilizando o EIP. Atualmente ela pode determinar três comportamentos.

1. Gravaria o registro ou não?
2. O programa deve iniciar EIP na próxima linha?
3. Iniciaria o EIP no início da linha ou na coluna corrente?

Estas três decisões são determinadas ao adicionar os EQUATES. Que são:

- EIPAction:Default Equate(0) !Salva de acordo com o método ASK
- EIPAction:Always Equate(1) !Salvar sempre as alterações
- EIPAction:Never Equate(2) !Nunca salvar as alterações
- EIPAction:Promped Equate(4) !Perguntar se salva as alterações
- EIPAction:Remain Equate(8) !Continue editando
- EIPAction:RetainColumn Equate(16) !Mantém a posição em uma nova linha

Por Exemplo:

Digamos que desejamos sempre salvar o registro e queremos que continue a edição em uma nova linha, mas na mesma coluna. Então faríamos assim:

Código:

```
Brwl.ArrowAction = EIPAction:Always + EIPAction:Remain +  
EIPAction:RetainColumn
```

Se você quiser que este comportamento seja alterado baseado na linha ou coluna selecionada, então o melhor lugar para colocar este código será no método ASK como descrito na propriedade AskProcedure. Contudo, recomenda-se que seja colocado no método INIT do objeto THISWINDOW.

EnterAction – Propriedades

Esta propriedade marcará a ação se o usuário pressionar a tecla ENTER, enquanto estiver editando com EIP. Esta propriedade governa ambas as ações que fará para salvar o registro corrente e também se continuará no modo EIP na linha abaixo. Isto quer dizer que pode fazer da tecla ENTER a tecla GRAVAR e ainda manter-se em modo de edição. Alternativamente agir igual em uma planilha que ao pressionar a tecla ENTER grava e sai do modo de edição. Os EQUATES usados são:

```
EIPAction:Default Equate(0) !Salva de acordo com o método ASK  
EIPAction:Always Equate(1) !Salvar sempre as alterações  
EIPAction:Never Equate(2) !Nunca salvar as alterações  
EIPAction:Promped Equate(4) !Perguntar se salva as alterações  
EIPAction:Remain Equate( 8 ) !Continue editando
```

Por exemplo:

Para salvar a célula e cessar o modo de edição:

Código:

```
Brwl.EnterAction = EIPAction:Always
```

Ou para salvar a célula e continuar no modo de edição:

Código:

```
Brwl.EnterAction = EIPAction:Always + EIPAction:Remain
```

Se quiser que este comportamento seja alterado conforme o conteúdo da linha ou coluna corrente, o lugar para colocar este código será no método ASK da classe Browse como descrito na propriedade AskProcedure. É recomendado que o comportamento não se altere, então o melhor lugar é colocar no método INIT do objeto THISWINDOW.

TabAction – Propriedades

Esta propriedade trabalha exatamente como a EnterAction, exceto pelo motivo de que se aplica quando o usuário pressionar a tecla TAB e não a tecla ENTER.

FocusLossAction – Propriedades

Esta é realmente uma propriedade interessante. Ela define o que deve ser feito se o controle de edição perder o foco enquanto estava alterando um campo. O registro deve ser gravado ou não? Você é quem deve decidir. Como nas propriedades anteriormente descritas, está também se utiliza dos EQUATES EIPAction. Menos, é claro, se manter em edição. Os EQUATES usados são:

```
EIPAction:Default Equate(0) !Salva de acordo com o método ASK  
EIPAction:Always Equate(1) !Salvar sempre as alterações
```

EIPAction:Never Equate(2) !Nunca salvar as alterações
EIPAction:Promped Equate(4) !Perguntar se salva as alterações

Por exemplo:

Código:

```
Brwl.FocusLossAction = EIPAction:Always
```

Se você quiser que este comportamento seja baseado no conteúdo do registro da linha ou coluna corrente, então o lugar para colocar o código é NewSelection, muito embora recomenda-se manter um único comportamento e então o melhor lugar será em INIT da THISWINDOW.

Em primeiro lugar, não pense que esta é uma ação que ocorre quando a Window perde o foco. Quando a Window perde o foco nada acontece. Ela simplesmente restaura o modo de edição quando o usuário retorna. Esta propriedade determina o que acontece quando o controle de edição perde o foco para um outro controle dentro da própria Window. Por exemplo, vamos supor que enquanto esta no modo de edição o usuário pressiona o botão FECHAR. Então a ação determinada é ativada.

AddEditControl – Método

Neste método é onde você pode especificar qual EditClass que será utilizado por uma coluna no Browse. Lembre-se de que cada coluna pode ser editada de um jeito diferente, algumas podem ser DROP-DOWN, outras ENTRY ou ainda CHECKBOX e por aí vai. Se não for indicada estas funções para as colunas, então automaticamente será empregada a classe EditEntryClass.

Este método leva três parâmetros. O primeiro marca o EditClass a ser empregado. Elas são configuradas no objeto que você tem de criar baseadas no EditClass. Por exemplo, se você quiser usar o EditEntryClass primeiro é necessário criar um objeto. Vamos chamá-lo de COLUMN1 e é do tipo EditEntryClass.

Código:

```
Column1 EditEntryClass
```

Então a chamada usaria o objeto COLUMN1 como o primeiro parâmetro.

O segundo parâmetro é o número da coluna. O terceiro é muito avançado que não precisamos discutir agora, ele pode ser omitido. Então nosso exemplo ficaria assim:

Código:

```
Brwl.AddEditControl(COLUMN1, 1)
```

Se omitir o primeiro parâmetro então ele é considerado como sendo label e não é editado sob hipótese alguma. Nós utilizaríamos isto se nossa coluna fosse do tipo calculada, como um total, ou se pertence a um arquivo secundário. Por exemplo:

Código:

```
Brwl.AddEditControl(, 2)
```

Isto quer dizer que a coluna 2 não pode ser editada.

Editando o Registro

Bem, nosso browse está pronto. Indicamos todas as ações padrões, identificamos todas as colunas editáveis e então vamos ver o que acontece quando o usuário edita um registro.

Como o usuário entra no modo de edição?

Enunciamos nesta matéria que o LISTBOX pode se comportar como uma planilha. Então o mais básico a fazer é marcar o atributo "SELECT COLUMNS" do LISTBOX. Isto também facilita muito para o usuário final ver qual coluna está selecionando. Marque este atributo em qualquer uma nas propriedades do LISTBOX do editor de WINDOW ou via código conforme

abaixo. Colocando-o no método INIT do objeto THISWINDOW logo após o comando OPEN(WINDOW).

Código:

```
?Browse:1{prop:Column} = 1
```

Existe um bom número de formas para o usuário iniciar o modo de edição. Elas se aplicam se um FORM ou um EIP estiver habilitado.

Primeiro a classe BrowseClass automaticamente adiciona as teclas INSERT, DELETE e CTRL+ ENTER para incluir, excluir e alterar respectivamente. Também o duplo clique com o botão esquerdo do mouse que significa alterar o registro selecionado. Ela também marca o botão ALTERAR para ser acionado como DEFAULT na WINDOW. Então se o usuário pressionar a tecla ENTER entra automaticamente em modo de alteração. Contudo, às vezes você quer que a tecla ENTER signifique SELECIONAR e não ALTERAR. (Quando o browse tem o comportamento de LOOKUP).

Segundo, note que na ordem para incluir, alterar e excluir registros os botões INCLUIR, ALTERAR e EXCLUIR são requeridos. Estes botões não precisam ser visíveis (eles podem estar invisíveis), mas são requeridos.

Neste ponto, eu mencionaria uma anormalidade. Se o usuário final usar o botão do mouse para entrar no modo de edição então a edição imediatamente vai para a coluna que estava apontando. Se o atributo SELECIONAR COLUNA estiver ligado e o usuário usar a tecla ENTER para entrar no modo de edição então o programa inicia a edição na coluna de número 1. O pequeno código abaixo descrito cuida desse problema.

Se você deseja que a tecla ENTER se comporte como o duplo-clique do mouse então adicione o seguinte código no método BRW1::EIPManager.INIT, antes do "parent call".

Código:

```
If Self.ListControl{prop:column}>0 then  
  Self.Column = Self.ListControl{Prop:Column}  
End
```

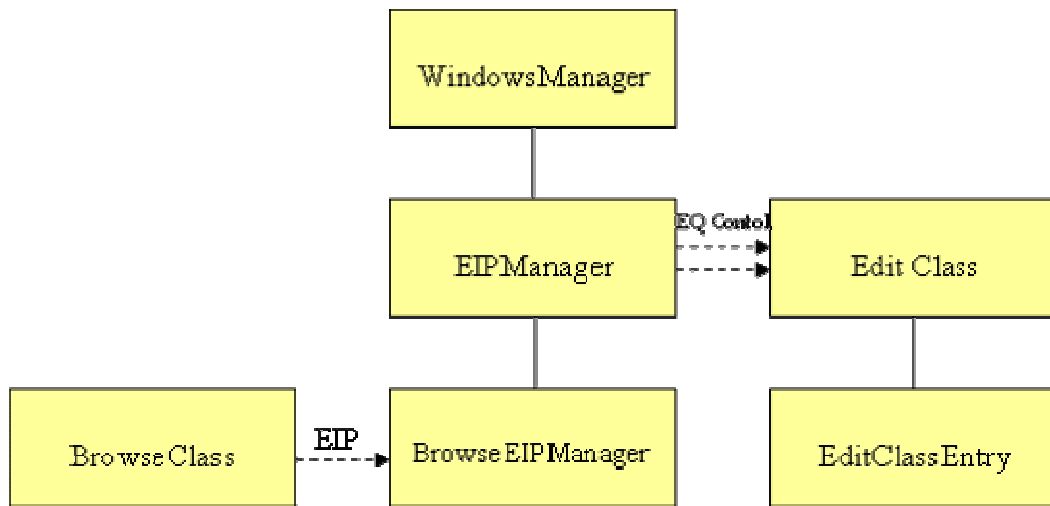
Fluxo geral superficialmente seria ...

- WindowManager.INIT - marque a propriedade da coluna no LISTBOX
- BrowseClass.Init - marque as ações de gravação
- BrowseClass.AddEditControl - adicione controles especiais da edição
- WindowManager.TakeEvent - evento dentro do loop accept
- BrowseClass.TakeEvent - eventos destinados ao browse
- BrowseClass.Ask - decide se chama EIP ou FORM, se for EIP então
- BrowseClass.AskRecord - chama o método RUN dos objetos EIP
- BrowseClass.EIP.Run

Isto nos trás para a propriedade de EIP do BrowseClass. Lembre-se de que o EIP é realmente um objeto baseado na classe BrowseEIPClass.

Classe BrowseEIPManager

Neste ponto vamos refrescar nossa memória com uma grande imagem.



Então a classe BrowseEIPManager é baseada na classe EIPManager que por sua vez é baseada na classe WindowsManager. Se olhar para o diagrama acima, verá que a classe EIPManager provém as funcionalidades básicas do EIP. A classe BrowseEIPManager atua apenas ligando a classe EIPManager e a classe BrowseClass.

A classe BrowseClass chama a classe BrowseEIPManager pelo método RUN. Uma rápida olhada nas declarações revela que não existe método RUN na classe BrowseEIPManager há somente um pequeno método na classe EIPManagerClass. O método RUN simplesmente guarda a ação requerida e então chama normalmente o método RUN da classe WindowManager. Isto quer dizer que o método RUN da WindowManager é usado e devemos lembrar que o método RUN primeiro chama o método INIT então o ASK e depois o KILL. Primeiro vamos olhar em detalhe o método INIT.

BrowseClass.Init - marque as ações de gravação
 BrowseClass.AddEditControl - adicione controles especiais da edição

Neste ponto a Window do Browse é aberta com todos os recursos. Naturalmente a WindowManager.TakeEvent é quem a acionou.

WindowManager.TakeEvent - evento dentro do loop accept
 BrowseClass.TakeEvent - eventos destinados ao browse
 BrowseClass.Ask - decide se chama EIP ou FORM, se for EIP então
 BrowseClass.AskRecord - chama o método RUN dos objetos EIP

Neste ponto o usuário seleciona uma “célula” e está pronto para iniciar uma edição mas, primeiro a inicialização do EIP precisa ocorrer.

BrowseClass.EIP.Run (EIPManager.Run)
 WindowManager.Run
 BrowseEIPManager.Init
 EIPManager.Init
 EIPManager.InitControls - Inicializa todos os controles e adiciona onde necessário for.

Agora o usuário está pronto para poder iniciar a entrada de valores em uma “célula”. A seqüência seguinte é para quando o usuário deixa uma célula (pressionando uma tecla ou, clicando no mouse ou ainda porque o programa perdeu o foco).

WindowManager.Ask
 EIPManager.TakeEvent

EIPManager.TakeFocusLoss
EIPManager.TakeCompleted
BrowseEIPManager.TakeCompleted - Isto grava o registro
WindowManager.TakeEvent
WindowManager.TakeWindowEvent
BrowseEIPManager.TakeNewSelection - Clicou em um outro registro
EIPManager.TakeFocusLoss - A Windows apenas perdeu o foco
BrowseEIPManager.TakeCompleted - Grava o Registro
EIPManager.TakeFieldEvent
EIPManager.TakeAction - O usuário pressionou alguma tecla aceleradora
BrowseEIPManager.TakeCompleted - Grava o registro
EIPManager.TakeFocusLoss - A Window apenas perdeu o foco
BrowseEIPManager.TakeCompleted - Grava o registro

Provavelmente ainda não está fazendo muito sentido, mas vamos olhar para a classe EditClass e então uniremos as três partes e assim sairemos com alguma coisa mais útil.

As Classes Edit – Uma primeira olhada

Como mencionamos anteriormente, cada coluna no browse pode ter um comportamento diferente. Estas diferenças são porque existe um número de classes que juntas são chamadas de classes editoras (Classes Edit). Todas estas classes abrangem as mesmas funcionalidades, todas têm os mesmos métodos e todas são baseadas na classe EditClass.

Incidentalmente a própria classe EditClass não é suficiente, uma derivação com a classe EditEntryClass é requerida. O código para estas classes é ativado em três posições.

Primeiro quando o “browse window” é inicializado, então todos os controles EIP são registrados. O template BROWSE faz isso geralmente por uma chamada para AddEditControl, dentro de método INIT do objeto BRW1, para cada campo EIP. Isto, por sua vez, chama o método EIPManager.AddControl. Um ponteiro no template gerado na EditClass é incluído na chamada para AddEditControl.

```
BrowseClass.AddEditControl  
BrowseClass.CheckEIP  
BrowseClass.BrowseEIPManager.AddControl
```

Depois deste objetivo a classe de edição é dobrada.

Primeiro naturalmente o INIT e KILL com suas funcionalidades. Isto inclui criar o controle o qual será usado para a edição e os alertas para as teclas de funções. Nós já anotamos a seqüência de inicialização, mas de qualquer modo:

```
BrowseClass.EIP.Run (EIPManager.Run)  
WindowManager.Run  
BrowseEIPManager.Init  
EIPManager.Init  
EIPManager.InitControls - Inicializa todos os controles e adiciona onde necessário  
for.
```

EditClass.Init

EditClass.CreateControle

EditClass.SetAlerts

Segundo, os eventos para o controle. Na seqüência abaixo vemos como a `EIPManager.TakeFieldEvent` chama o método `EditClass.TakeEvent`.

`EIPManager.TakeFieldEvent`

EditClass.TakeEvent

`EIPManager.TakeAction` - O usuário pressionou alguma tecla aceleradora

`BrowseEIPManager.TakeCompleted` - Grava o registro

`EIPManager.TakeFocusLoss` - A Window apenas perdeu o foco

`BrowseEIPManager.TakeCompleted` - Grava o registro

De qualquer forma a classe `EditClass` é a mais simples das três partes e então não há muito o que se possa fazer. O controle é criado no método `CreateControl` (o qual dissemos anteriormente que ocorre durante a fase `INIT`). O LABEL equivalente do novo controle é colocado nas propriedades `FEQ`. As setas, tecla `TAB` e tecla `ENTER` são assim alertadas no método `SetAlerts`.

EditEntryClass

Esta classe serve somente para criar o corretamente o controle `ENTRY` para a célula quando for preciso editar. Esta é a classe padrão a ser utilizada por todos os campos do browse se nenhuma outra classe for especificada.

EditCheckClass

Esta classe, como a `EditEntryClass` serve somente para criar o controle `CHECKBOX` para edição.

EditSpinClass

Esta classe cria o controle `SPIN` e também estabelece os limites superior e inferior para ele.

EditDropListClass

Esta classe margeia a classe `EditEntryClass`, mas um pouco mais complicada. Ela cria o controle, mas também sobre-escreve o método `SetAlerts`, simplesmente as setas para cima e para baixo não podem mais alertar o controle.

Agora nós cobriremos todas as bases, pelo menos em um nível introdutório, vamos ver o que podemos fazer com tudo isso.

Embedding no EIP

Talvez a melhor forma de conhecermos a fundo os lugares mais indicados é quando geramos um procedimento e adicionamos algum código e percebemos o que podemos fazer.

Condicionalmente vá para o EIP ou para o FORM

Já vimos isto anteriormente utilizando a propriedade `AskProcedure`.

Inicializando os campos quando incluindo um registro

Inicializar é talvez a coisa mais complicada que nós precisaremos ver, embora seja um tanto inesquecível, se você começar errado, mas é muito seguro se você souber o que está fazendo.

Primeiro vamos olhar para o `PRIME` de um registro quando estiver numa inclusão.

Existem vários locais onde poderíamos colocar este código, mas o mais elegante é colocá-lo no método `PrimeRecord` da classe `BrowseClass`. Você precisa colocar seu código depois do "parent call". Os campos para `PRIME` são do arquivo original ou variáveis de memória, mas não variáveis `QUEUE`.

Outro lugar que podemos inicializar alguma coisa é no método INIT da classe BrowseElPManager. Novamente depois do "parent call" seja o melhor lugar. Entretanto este não é o melhor lugar para iniciar valores dos campos, pois eles já foram copiados para as variáveis da QUEUE e terão dificuldades para mostrar seus valores imediatamente, mas se você precisar fazer alguma rotina para não inicializar algum campo este é o melhor lugar.

Deve-se evitar utilizar o método PRIMEFIELDS da classe BrowseElPManager. Este método não é chamado durante o processo de iniciação. Trata-se de um resto da WindowManager do qual o BrowseElPManager é derivado.

Quando incluir um registro você pode iniciar os campos usando o método PrimeRecord (BRW1.PrimeRecord) da classe BrowseClass. Depois do "parent call".

Selecionando uma coluna para edição

Dissemos anteriormente que se o usuário pressionar a tecla ENTER iniciará a edição de uma célula. Ele automaticamente é movido para a primeira coluna da linha, contudo se a propriedade PROP: COLUMN do LISTBOX estiver marcada, então ela muito provavelmente não quer iniciar a edição na coluna selecionada.

A propriedade aqui para editar pertence a classe ElPManager, ela é denominada COLUMN. Ajustando isto durante o processo de inicialização você pode iniciar a edição em uma coluna específica.

Exemplo

Se você quiser o comportamento da tecla ENTER seja o mesmo do DUPLO-CLIQUE então adicione o seguinte código para o método "BRW1::ElPManager.Init", antes do "parent call".

Código:

```
If Self.Reg = ChangeRecord and Self.Column = 0 then
    Self.Column = Self.ListControl{Prop:Column}
End
```

Observe que nós precisamos verificar duas coisas. Primeiro verificar se estamos no modo de alteração e também necessitamos verificar o estado corrente da propriedade COLUMN porque nós somente desejamos executar este código quando iniciar a edição, sob pena de interferir com o funcionamento das setas para cima ou para baixo.

Marcando a coluna selecionada no browse para seguir a coluna editada

Existe uma parte de código que faz com que a coluna inteira trabalhe adequadamente. O que nós queremos fazer aqui é pegar a coluna selecionada do browse para seguir a coluna que estava sendo editada. Em outras palavras quando a edição estiver completa queremos que a última coluna do browse seja a selecionada.

O código para isso pertence ao método TakeEvent da classe BrowseElPManager antes do "parent call".

Código:

```
If event() = Event:Selected and KeyCode() <> MoseLeft and
KeyCode() <> RightMouse then
    Self.Listcontrol{Prop:Column} = Self.column
End
```

O truque é desconsiderar os botões do mouse.

Quando a edição da coluna está concluída

Este é um ponto embutido muito popular e útil, mas também é muito difícil de encontrar. Num primeiro instante, você procura por alguma coisa no evento ACCEPTED, mas ele simplesmente não existe. O que desejamos aqui é inspecionar como foi a entrada, por questões de validação ou mesmo para corrigi-la ou ainda iniciar valores para outros campos do registro.

Depois de muito tempo inspecionando, achei que o melhor ponto aqui é o método TakeEvent da classe BrowseEIPManager (BRW1::EIPManager.TakeEvent) antes de chamar o "parent call". Precisamos fazer uma pequena verificação do evento correto e também checar a coluna que foi editada.

Para adicionar códigos apenas depois que cada coluna foi editada, use o método TakeEvent da classe BrowseEIPManager (BRW1::EIPManager.TakeEvent) antes de chamar o "parent call".

Importante notar aqui é que não precisamos marcar a variável do arquivo. Você deve ajustar a variável QUEUE. Isto nos leva para um ponto interessante. Isto quer dizer que somente podemos carregar os campos que estiverem sendo mostrados no LISTBOX. Caso o desejo seja editar um campo que não esteja no LISTBOX, inclua-o nos HOT FIELDS do browse que ele será adicionado na QUEUE.

Exemplo:

Digamos que tem um pedido estilo browse. A coluna 3 contém a quantidade e a coluna 4 o desconto, sempre que a quantidade ou o desconto forem alterados, então o preço da mesma linha precisa ser calculado.

Código:

```
If Event() = Event:AlertKey or Event() = Event:Accepted then
  Case Self.Column
    Of 3 orof 4
      Update() - Atualiza os campos da QUEUE
        Queue:Browse:1.Lin:Total = (Queue:Browse:1.Lin:Preco * |
                                   Queue:
Browse:1:Lin:Qtde) * |
                                   (100 - Queue:
Browse:1:Lin:Desconto / 100)
      end
    End
```

Atualizando alguma coisa a mais na Window

Algumas vezes desejamos atualizar um campo na Window enquanto cada linha esta sendo editada. Por exemplo, se o usuário estiver entrando com os itens de um pedido, então desejamos atualizar o valor total do pedido a cada linha digitada.

O lugar para colocar código é exatamente o mesmo do exemplo anterior, no método TakeEvent da classe BrowseEIPManager antes de chamar o "parent call".

Pouco antes de o registro ser gravado

Como em um template FORM, o registro é salvo dentro do método TakeCompleted entretanto, é importante escolher o método TakeCompleted correto. Veja que neste momento nós não estamos falando à respeito da THISWINDOW.TakeCompleted mas sim da classe BrowseEIPManager método TakeCompleted. Para complicar ainda mais, a classe BrowseEIPManager tem dois métodos TakeCompleted. O único que queremos é o que tem o parâmetro (BYTE Force).

O que é muito importante considerar aqui é que não devemos utilizar os campos do arquivo, mas somente as variáveis da QUEUE. Isto nos leva para um interessante ponto. Isto quer dizer de que somente podemos editar aqui os campos que estiverem presentes dentro da QUEUE do browse. Caso haja algum campo a manipular que não esteja presente, basta incluí-lo em HOT FIELDS do Browse.

Fontes Bibliográficas:

Help CLARION

Apostila ABC – Bruce Jonhson

Revista Clarion Magazine

Revista News Clarion Brasil