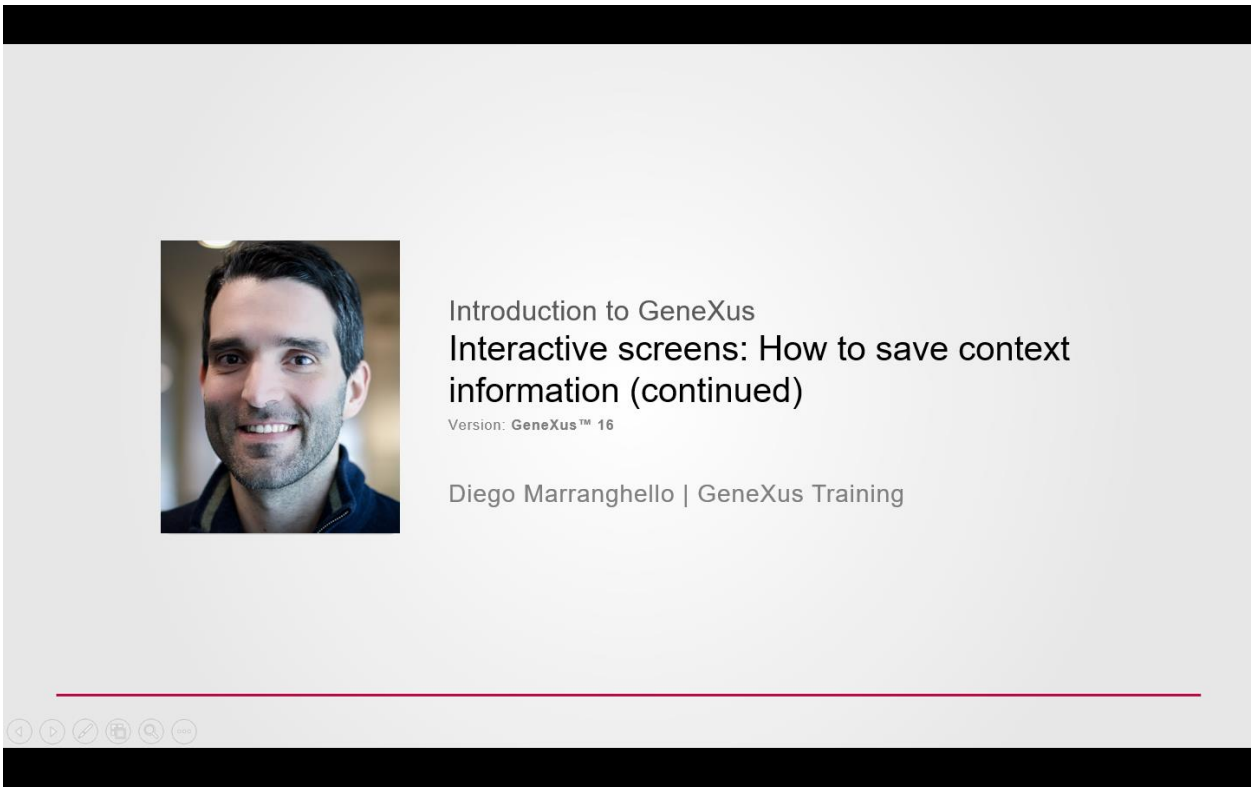
Telas interativas

Como salvar informações de contexto (continuação)



No vídeo anterior, vimos como manter dados em memória, evitando que sejam perdidos após chamar outro objeto e depois retornar ao objeto chamador.

Para isto, o que fizemos foi criar uma variável do tipo WebSession, e salvamos nela os dados que nos interessavam, que no nosso caso eram os valores dos três filtros disponíveis.

Decidimos salvar cada um destes dados no evento &Update.Click usando o método SET da variável de sessão, para recuperá-los posteriormente no evento Start, usando o método GET de nossa variável WebSession.

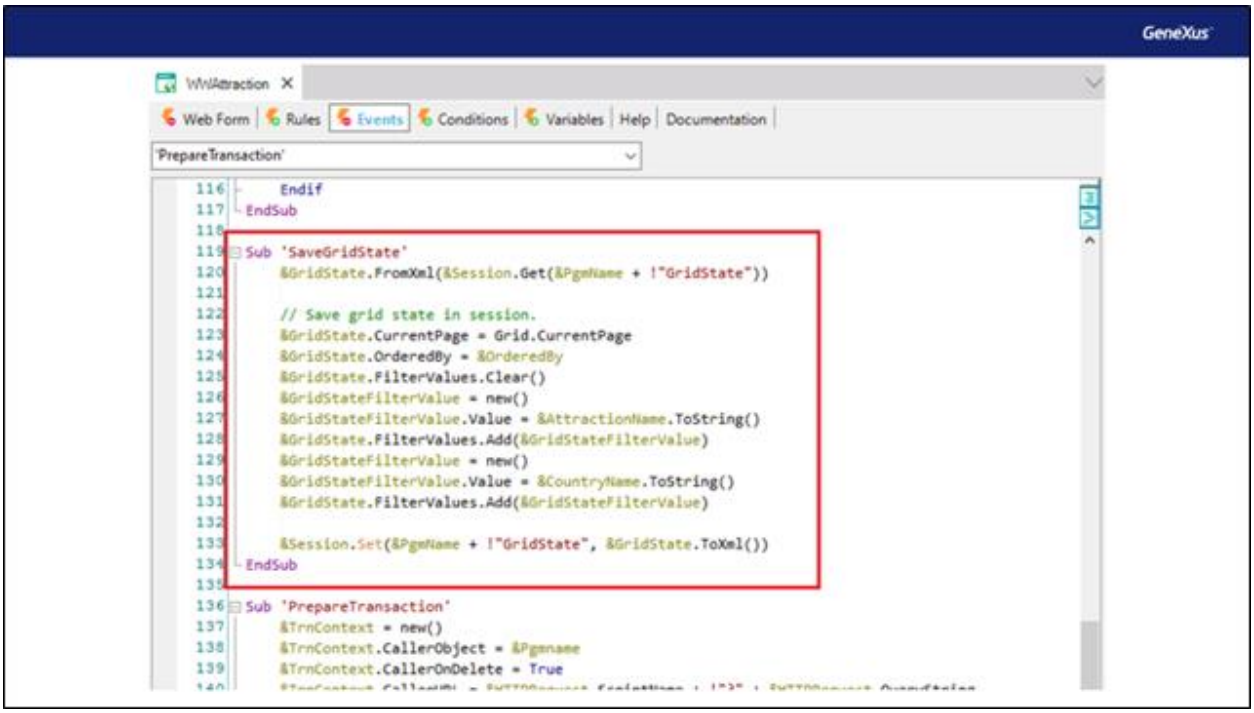
```
Event Load
    &trips = Count(TripDate)
    &totalTrips = &totalTrips + &trips
Endevent

Event Refresh
    &totalTrips = 0
Endevent

Event Start
    &update.FromImage(updateIcon)
    &CountryId = &webSession.Get('CountryId').ToNumeric()
    &AttractionNameFrom = &webSession.Get('AttractionNameFrom')
    &AttractionNameTo = &webSession.Get('AttractionNameTo')
Endevent

Event &update.Click
    &webSession.Set('CountryId', &CountryId.ToString())
    &webSession.Set('AttractionNameFrom', &AttractionNameFrom)
    &webSession.Set('AttractionNameTo', &AttractionNameTo)
    Attraction(trnMode.Update, AttractionId)
Endevent
```

Esta foi uma solução que criamos à nossa maneira e, como prometemos no vídeo anterior, veremos agora como o faz automaticamente o Pattern Work With da transação Attraction

Lembremos que a partir deste Work With gerado automaticamente pelo GeneXus, quando inserimos valores nos filtros e, em seguida, atualizamos algum registro a partir da ação Update, ao retornar os valores são mantidos. Vejamos como ele faz isso.

Vamos acessar o objeto WWAttracion e, em seguida, a seção de eventos dele.

Vemos que há uma sub-rotina gerada chamada “SaveGridState”



Uma sub-rotina é um bloco de código identificado com um nome, que poderá ser executado dentro do mesmo objeto.

As sub-rotinas são definidas com o comando Sub.

Uma das vantagens das sub-rotinas é que torna o código mais claro, facilitando sua leitura.

Outra grande vantagem é que permitem reutilizar o bloco de código declarado dentro. Desta forma, se precisamos executar o mesmo bloco de código de vários locais do objeto, se escreve apenas uma vez e pode ser chamado de vários locais utilizando o comando DO, veremos isto em um momento.

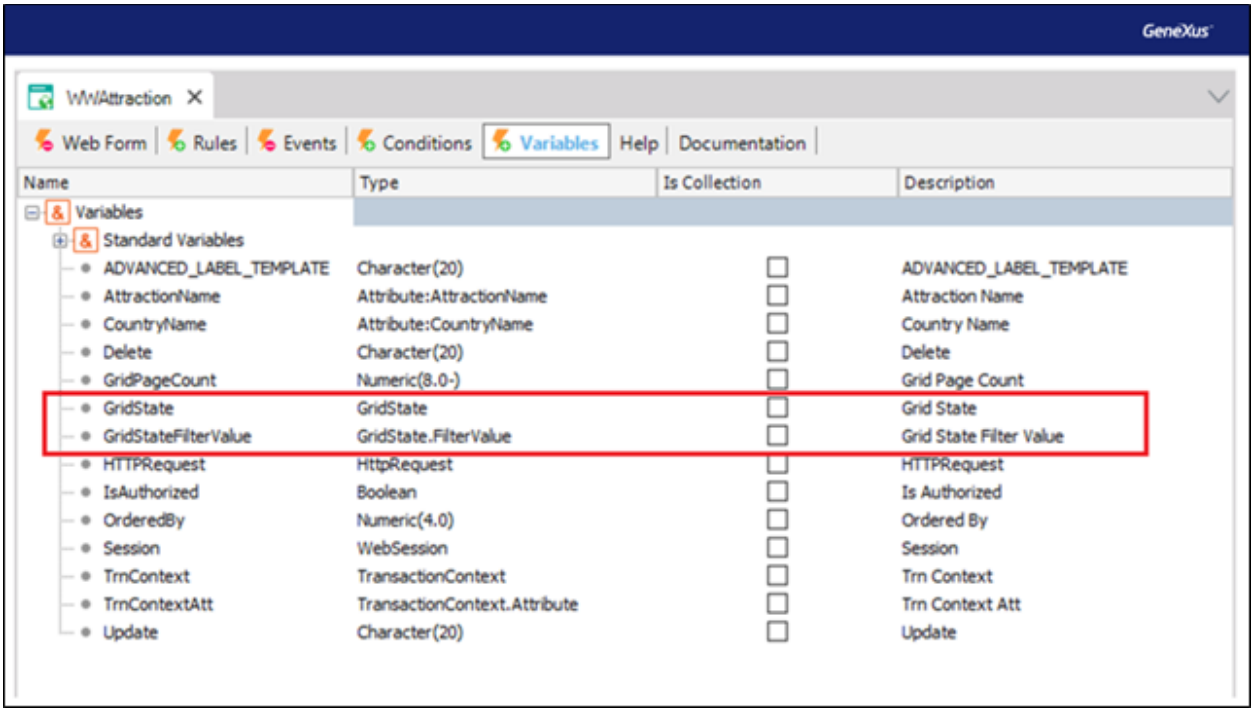
Vamos nos concentrar na seguinte seção do código desta sub-rotina, que é o que nos interessa neste caso.

GeneXus

```
// Save grid state in session.
&GridState.CurrentPage = Grid.CurrentPage
&GridState.OrderedBy = &OrderedBy
&GridState.FilterValues.Clear()
&GridStateFilterValue = new()
&GridStateFilterValue.Value = &AttractionName.ToString()
&GridState.FilterValues.Add(&GridStateFilterValue)
&GridStateFilterValue = new()
&GridStateFilterValue.Value = &CountryName.ToString()
&GridState.FilterValues.Add(&GridStateFilterValue)

&Session.Set(&PgmName + !"GridState", &GridState.ToXml())
```

Vemos antes de tudo que existe uma variável de nome GridState e outra de nome GridStateFilterValue.

Observemos a partir da seção variáveis como foram declaradas.

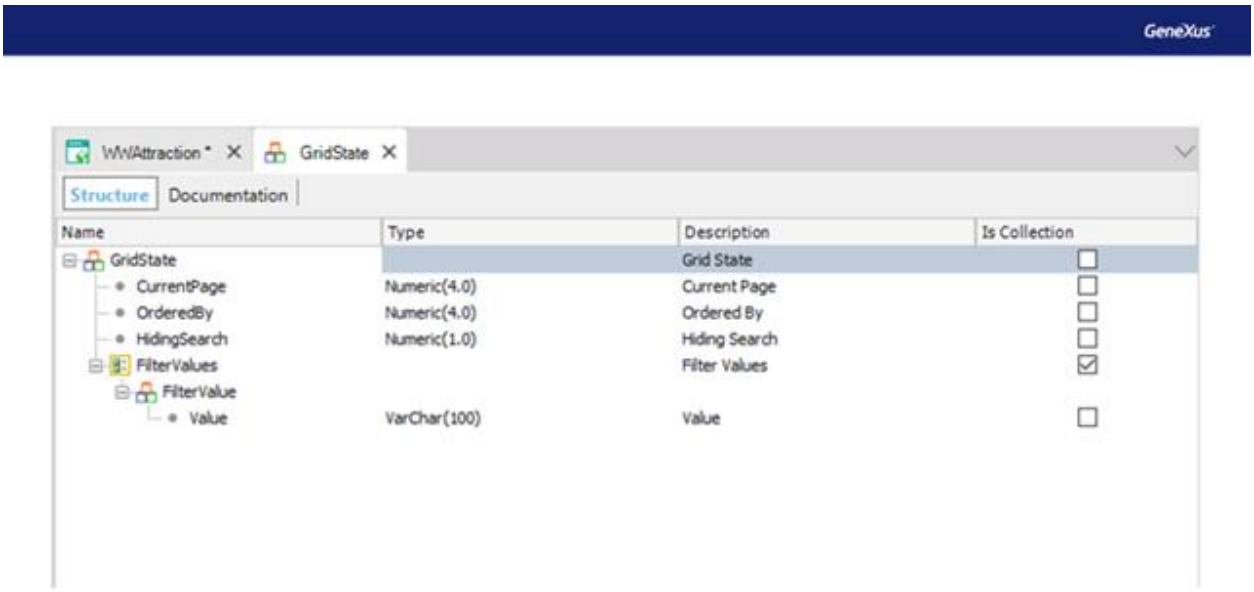


Vemos que GridState é do tipo GridState e GridStateFilterValue do tipo GridState.FilterValue.

Agora, a que se referem estes dois tipos?

Se procuramos dentro dos objetos de nossa KB, vemos que se refere a um objeto do tipo SDT, que foi gerado pelo GeneXus automaticamente ao criar o WorkWith.

Vejamos qual estrutura tem este SDT.



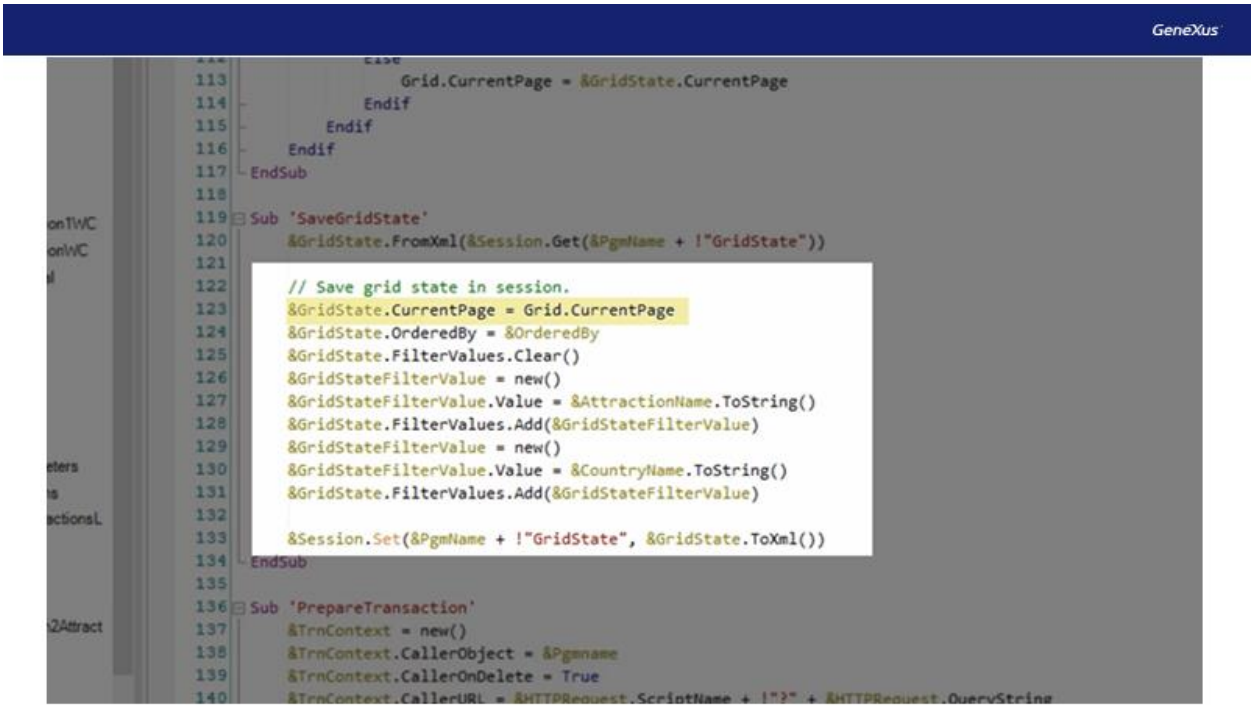
Vemos que há três membros declarados em sua estrutura principal, denominados CurrentPage, OrderedBy e HidingSearch. Os três do tipo Numérico.

Também vemos que existe uma subestrutura chamada FilterValue, a qual será uma coleção.

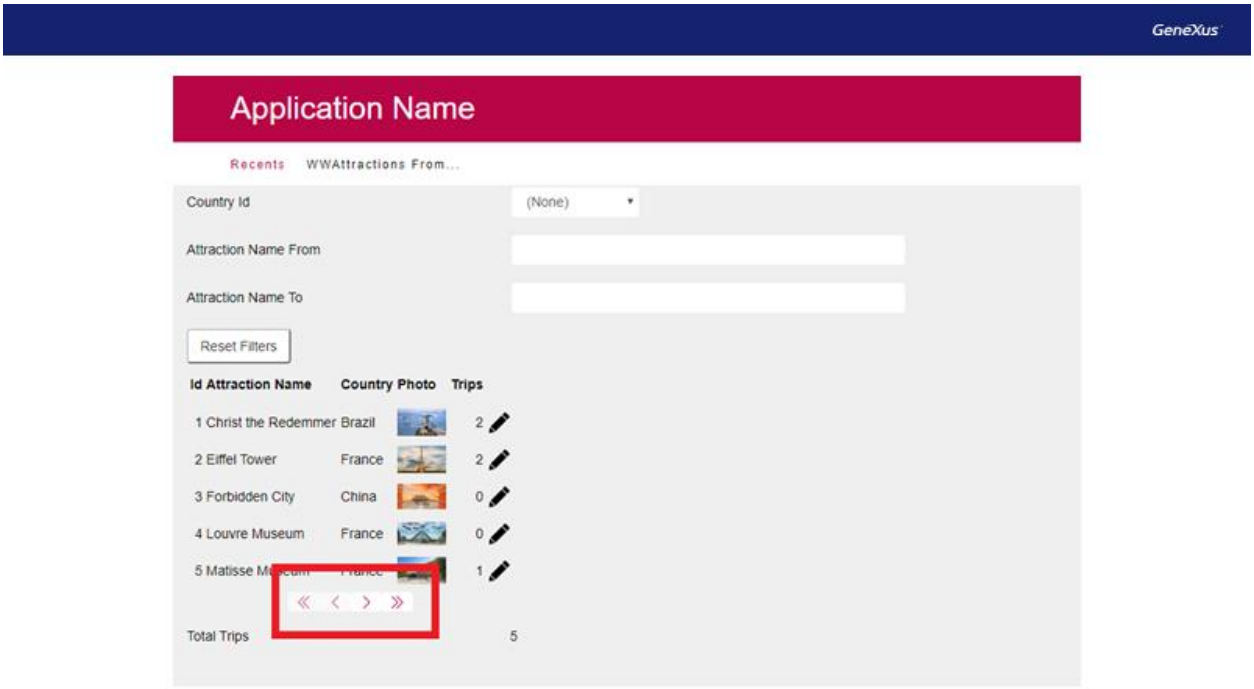
E dentro desta subestrutura existe declarado um membro chamado Value, do tipo Varchar.

Vamos voltar à seção eventos do objeto WWAttraction.

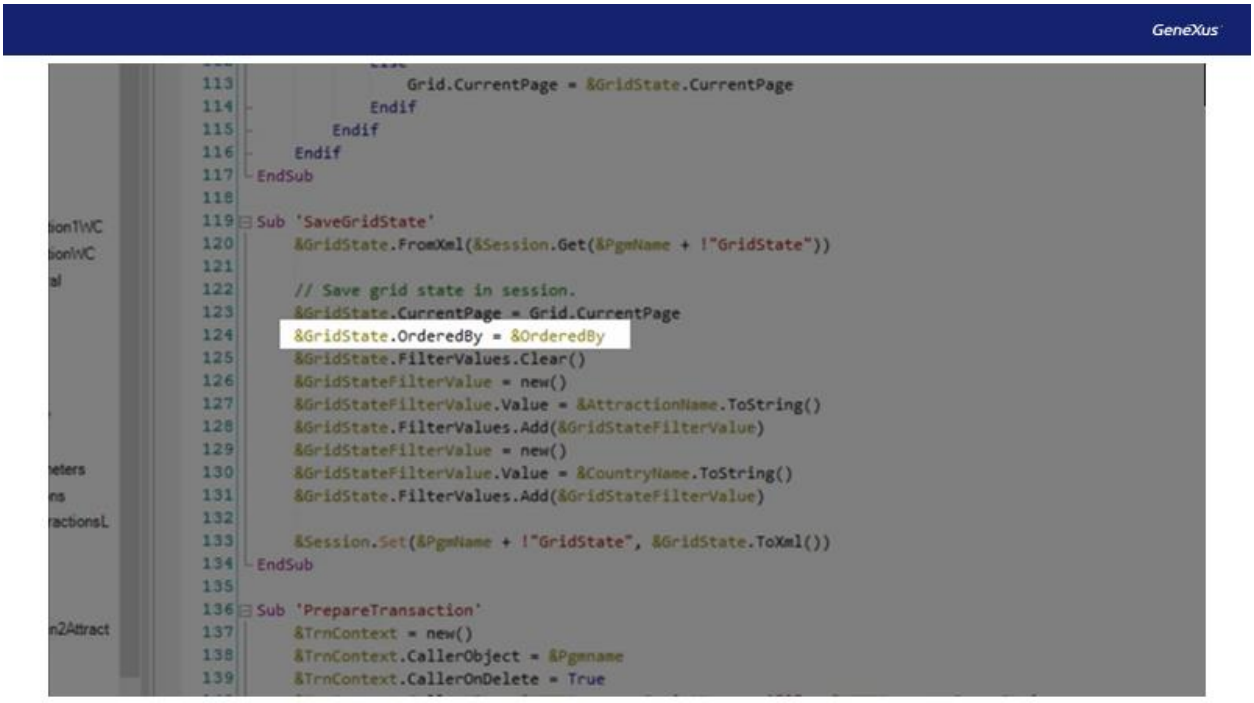
Dentro do código que nos interessa observar, vemos que a primeira linha guarda no membro CurrentPage da variável GridState, o número da página do grid na qual estamos.



No nosso caso, não tínhamos paginação no grid, mas, por exemplo, se em nosso WorkWith, configurássemos a propriedade Row do grid e colocamos um valor, por exemplo 5, aqui o que dizemos é que queremos mostrar 5 linhas por página.
Vemos ao atualizar que aparecerão apenas os 5 primeiros elementos e, em seguida, oferecerá a opção de avançar a página para observar os dados restantes. Nesse caso, também estaríamos interessados em salvar a página em que estávamos posicionados. É por isso que o Pattern faz isso, para cobrir esse cenário.

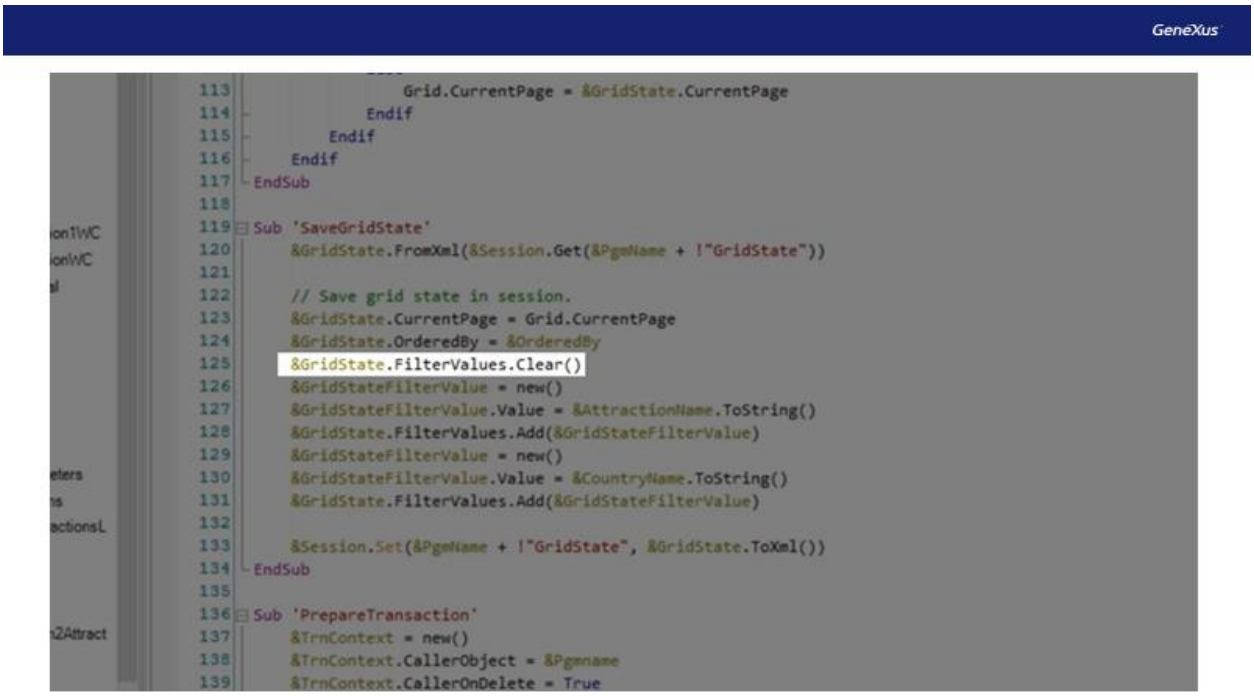


Na próxima linha será salva a ordem que pode ter sido aplicada aos dados, e serão salvas no membro OrderedBy do SDT GridState.



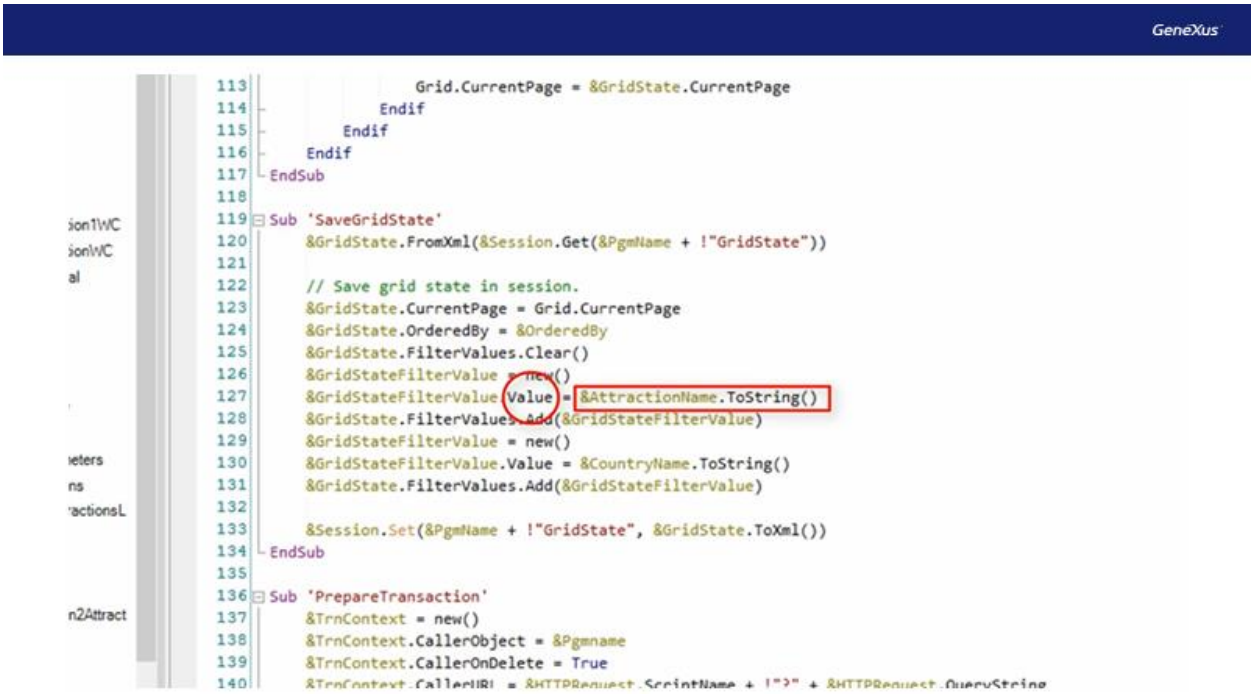
Em seguida, com o método Clear(), será excluído qualquer dado que tenha sido carregado na coleção FilterValues.

Agora aparecerá em ação a variável GridStateFilterValue, que como vimos, é do tipo GridState apontando para a subestrutura FilterValue.



Em primeiro lugar, é atribuído o operador new(), que retorna uma nova instância inicializada da variável SDT. Para mais tarde poder carregar um valor nela.

Isto é feito no membro Value, atribuindo a variável AttractionName, que é a utilizada pelo WorkWith para inserir o filtro por nome.



Em seguida, esta variável é adicionada à coleção FilterValues.

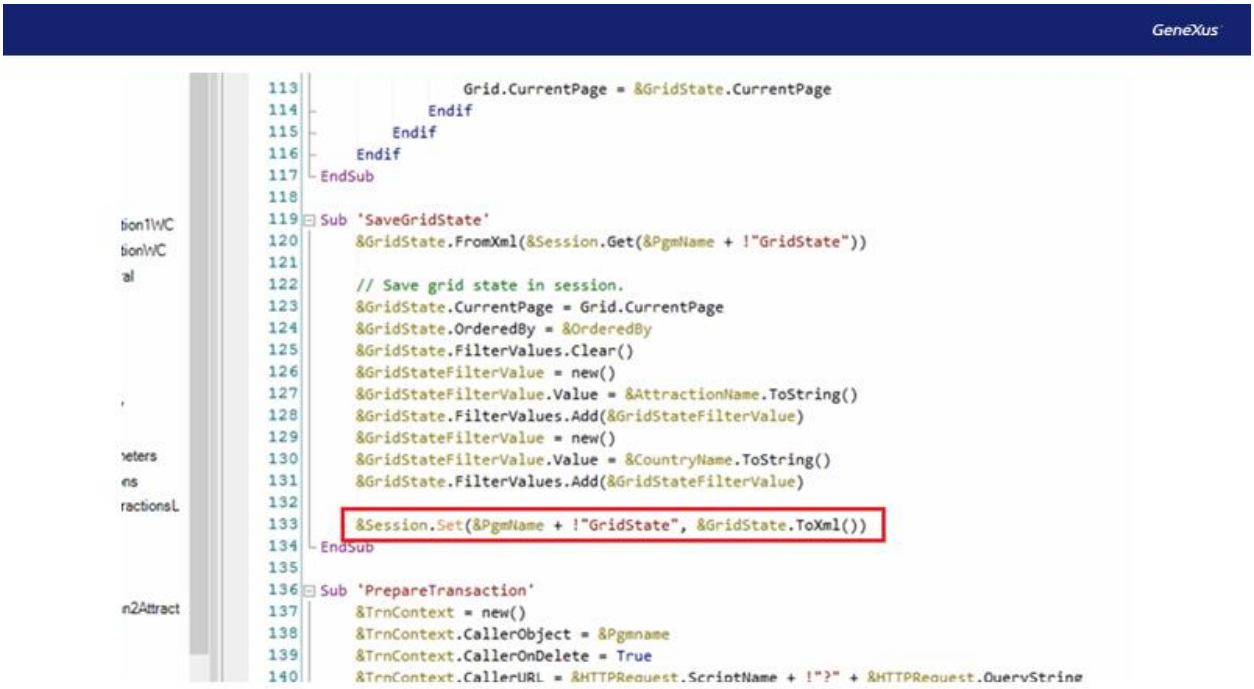
O mesmo procedimento é repetido novamente, mas neste caso com a variável CountryName, a qual conterá o valor do filtro por nome de país.

Observamos então que existe uma variável chamada Session, se formos para a seção variáveis, vemos que esta é do tipo WebSession.

Esta variável será a que guardará as informações coletadas nas linhas anteriores.

Para isto, será utilizado o método Set na variável de sessão, passando por parâmetros uma chave e um valor.

Como chave, utilizará o valor da variável PgmName concatenado com o texto GridState.



PgmName, é uma variável que armazena o nome do programa ativo. Ou seja, o nome especificado na propriedade name do objeto. Neste caso “WWAttraction”.

E como valor, salvará a variável GridState. Que como vimos, é do tipo SDT e é onde foram salvos os dados que deseja-se armazenar.

Vejamos agora, onde é feita a chamada para esta sub-rotina, ou seja, em que momento será utilizada.

Vemos que é chamada pelo comando “DO” no evento Refresh. Ou seja, toda vez que esse evento for disparado, serão salvos os valores dos filtros que interessam.

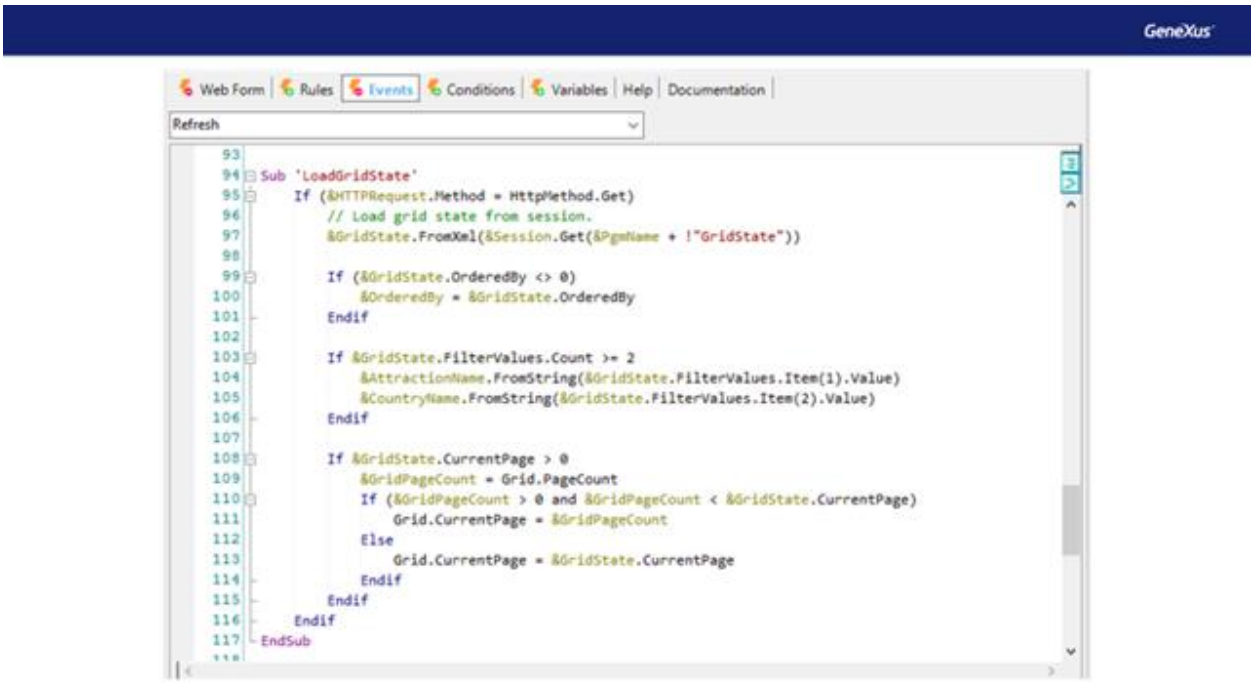
```
59          CountryNameFilterContainer.Class = ThemeClass:AdvancedContainerItem
60          &CountryName.Visible = false
61      endif
62  EndEvent
63
64  Event Refresh
65      Do 'SaveGridState'
66
67      do case
68          case &OrderedBy = 1
69              LblOrderBy.Caption = format(&ADVANCED_LABEL_TEMPLATE, "Ordered By", "Name")
70          case &OrderedBy = 2
71              LblOrderBy.Caption = format(&ADVANCED_LABEL_TEMPLATE, "Ordered By", "Country")
72      endcase
73
74      If &CountryName.IsEmpty()
75          LblCountryNameFilter.Caption = "Country Name"
76      Else
77          LblCountryNameFilter.Caption = format(&ADVANCED_LABEL_TEMPLATE, "Country Name", &CountryName)
78      Endif
79  EndEvent
80
81  Event Grid.Load
82      &Update.Link = Attraction.Link(TrnMode.Update, AttractionId)
83      &Delete.Link = Attraction.Link(TrnMode.Delete, AttractionId)
84      AttractionName.Link = ViewAttraction.Link(AttractionId, "")
85      CountryName.Link = ViewCountry.Link(CountryId, "")
86  EndEvent
87
```

No vídeo anterior, para a solução em nosso WebPanel, consideramos salvar os valores de nossos filtros neste evento. E decidimos que não era o melhor momento, porque toda vez que é disparado o evento Refresh, serão salvos estes valores, e talvez não tenhamos interesse em salvá-los, porque não precisaremos recuperá-los mais tarde e, no entanto, estamos salvando toda vez. Por exemplo, sempre que seja alterado o valor de um dos filtros, será disparado este evento e esses valores serão salvos.

O Pattern o faz desta maneira justamente porque é um padrão e, desta forma, contempla todas as situações que podem ocorrer, o pattern não sabe exatamente o uso que vamos dar à aplicação, é por isso que desta maneira cobre todos os cenários possíveis.

Bem, vejamos agora como e onde é que serão recuperadas estas informações que salvamos.

Observamos que foi gerada outra sub-rotina chamada LoadGridState. É aqui que serão recuperadas as informações armazenadas na sub-rotina SaveGridState.



Nesta linha, será carregada na variável GridState, que lembramos ser do tipo SDT, as informações salvas anteriormente na variável de sessão Session.

Isto é feito pelo método FromXml, que recebe uma string XML da qual são carregadas as informações no objeto SDT.

Para esse método, será passado por parâmetro de onde se deseja recuperar essas informações, que neste caso será da variável Session. Vemos que utiliza o método Get para indicar qual valor deseja recuperar usando a chave que foi salva o mesmo. Lembremos que a chave atribuída para salvar as informações foi a concatenação da variável PgmName, ou seja, o nome do objeto, mais a string de texto GridState.

Então vemos que são feitas várias verificações, usando os comandos IF.

```
    Attraction(TrnMode.Insert, nullvalue(AttractionId))
EndEvent

/*** Subroutines used to load and save the grid state. ***/

Sub 'LoadGridState'
    If (&HTTPRequest.Method = HttpMethod.Get)
        // Load grid state from session.
        &GridState.FromXml(&Session.Get(&PgmName + !"GridState"))

        If (&GridState.OrderedBy <> 0)
            &OrderedBy = &GridState.OrderedBy
        Endif

        If &GridState.FilterValues.Count >= 2
            &AttractionName.FromString(&GridState.FilterValues.Item(1).Value)
            &CountryName.FromString(&GridState.FilterValues.Item(2).Value)
        Endif

        If &GridState.CurrentPage > 0
            &GridPageCount = Grid.PageCount
            If (&GridPageCount > 0 and &GridPageCount < &GridState.CurrentPage)
                Grid.CurrentPage = &GridPageCount
            Else
                Grid.CurrentPage = &GridState.CurrentPage
            Endif
        Endif
    Endif
EndSub

Sub 'SaveGridState'
    &GridState.FromXml(&Session.Get(&PgmName + !"GridState"))
```

Nesta seção, verifica-se que o membro OrderedBy desta variável SDT seja diferente de 0, ou seja, verifica se há alguma ordenação salva, para saber se é necessário recuperá-la ou não.

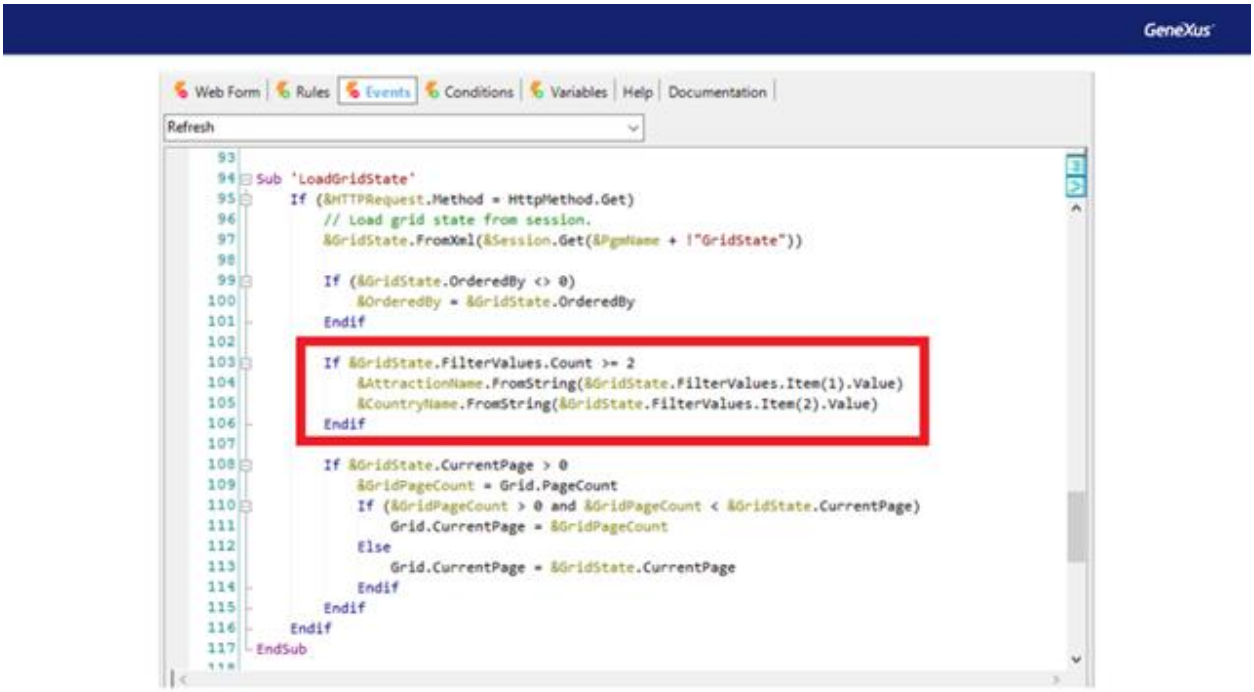
Se for diferente de 0, recupera as informações armazenadas em uma variável chamada OrderedBy.

GeneXus

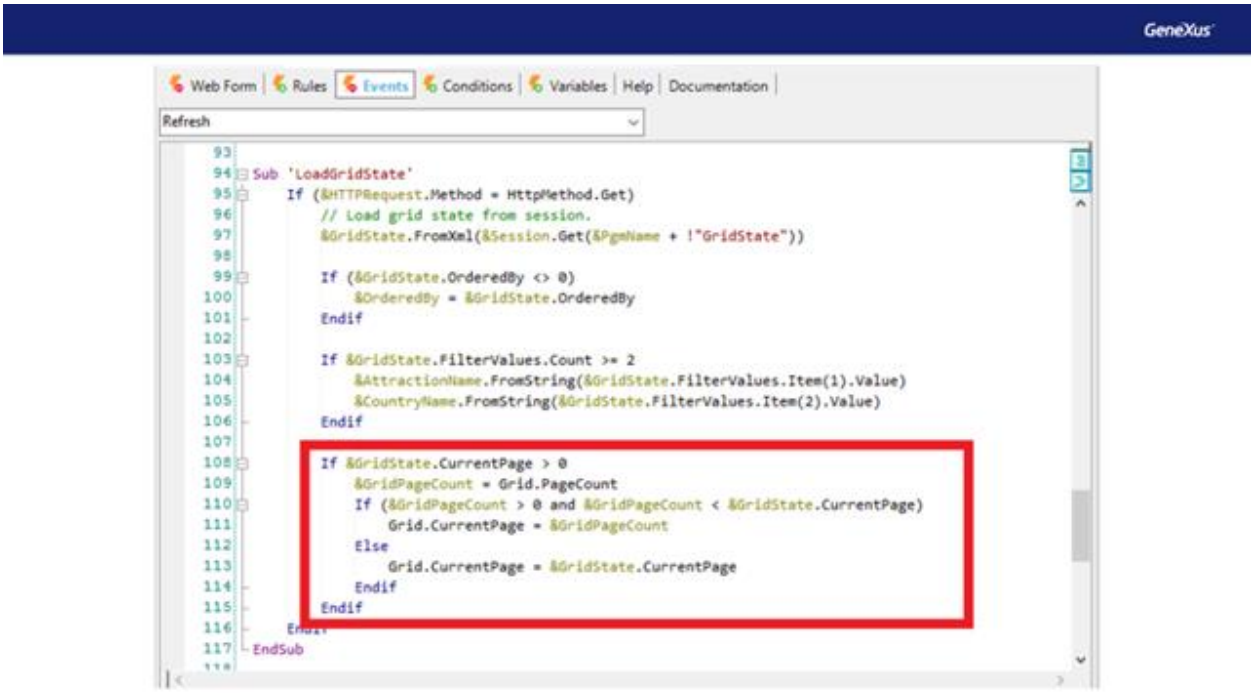
Em seguida, verifica se a quantidade de dados salvos na coleção filterValues é maior ou igual a dois. Este valor é atribuído porque são dois os filtros que existem no WorkWith. Verifica que estão salvos estes dados para posteriormente recuperá-los.

Neste bloco de código é onde os valores dos filtros serão recuperados.

O filtro do nome da atração será salvo na variável AttractionName, usando o método FromString. O mesmo com o filtro do nome do país, salvando o valor na variável CountryName.

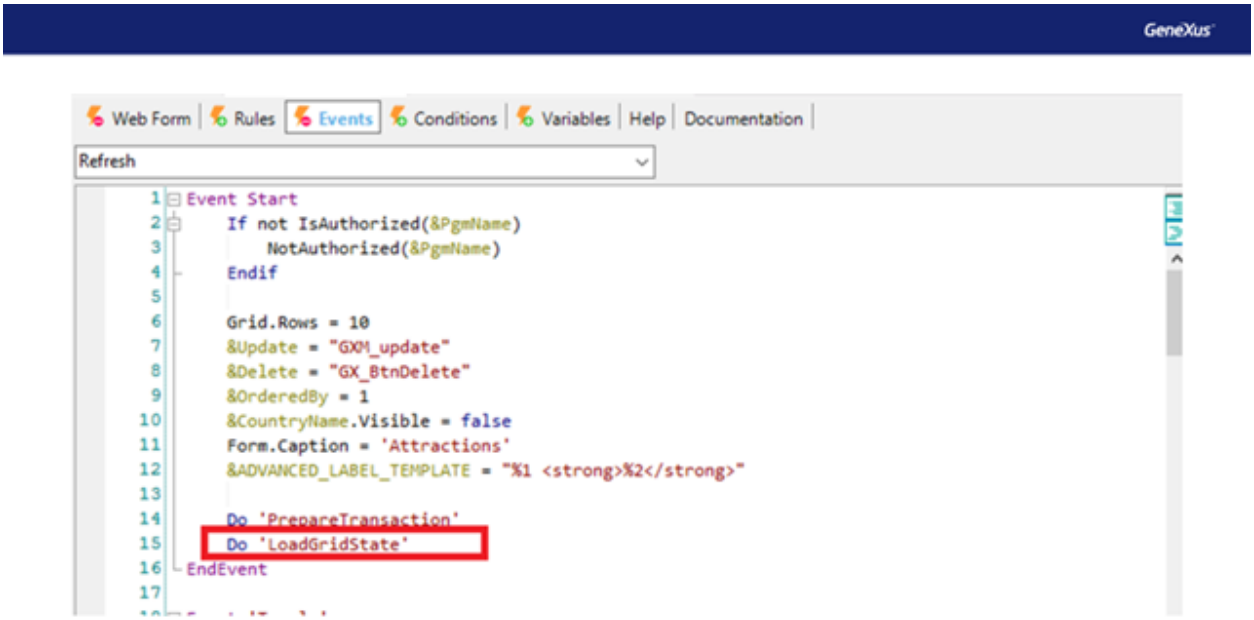


A última coisa verificada é se o membro CurrentPage, utilizado para salvar a página do grid, é maior que 0. Nesse caso, recupera este valor e o atribui diretamente à propriedade CurrentPage do grid.



Agora, quando será o momento que se chamará esta sub-rotina?

Como analisamos quando criamos nosso próprio Web Panel, o único momento possível será no evento Start. Evento que é executado apenas uma vez quando o site é carregado pela primeira vez.



Outra diferença que este objeto apresenta com o criado por nós, podemos ver no evento Load do grid.


```
80
81  Event Grid.Load
82      &Update.Link = Attraction.Link(TrnMode.Update, AttractionId)
83      &Delete.Link = Attraction.Link(TrnMode.Delete, AttractionId)
84      AttractionName.Link = ViewAttraction.Link(AttractionId, "")
85      CountryName.Link = ViewCountry.Link(CountryId, "")
86  EndEvent
87
```

Para acessar para atualizar as informações de algum registro do grid, tanto o nosso web Panel quanto o gerado pelo pattern, utilizam a variável &update, clicando nela nos leva à tela correspondente, que neste caso é a transação attraction em modo update.

Lembremos de como o implementamos em nosso Web Panel.

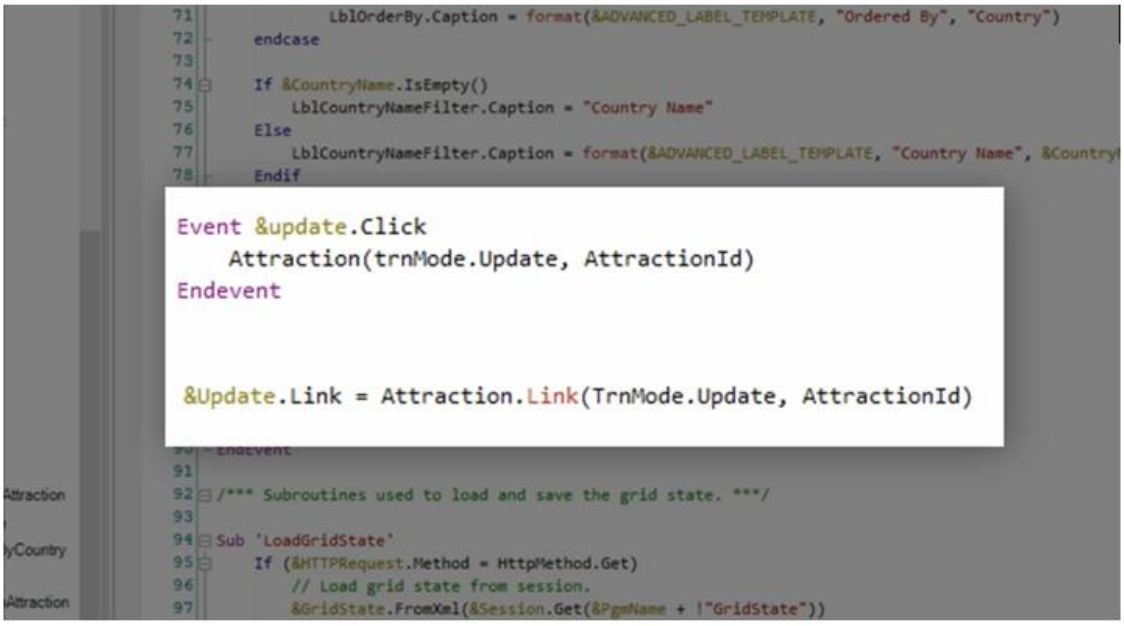
Fizemos isso dentro do evento &update.click, chamando diretamente a transação Attraction, passando por parâmetro o modo como queremos que seja executada e o id da atração.

O pattern, faz no evento Grid.Load, o qual será executado tantas vezes quantos forem os registros a serem carregados no grid.

Observemos que o Pattern aplicou a propriedade Link à variável Update. E então informa que esse valor será igual à função Link() da transação Attraction, passando por parâmetro o modo que se deseja executar essa transação e o ID que a identifica, neste caso AttractionId. Ou seja, associa a função Link à propriedade link da variável, resultando que ao fazer clique nessa variável, é realizada a chamada para o objeto web Attraction.

Embora não estejamos exemplificando, faz o mesmo com a variável &delete, passando por parâmetro o modo correspondente.

Neste caso, as duas formas de implementá-lo terão a mesma funcionalidade.
Quando clicamos na variável update, tanto em uma como na outra opção, será chamado o objeto Attraction, indicando que queremos realizar uma atualização, passando o Id do elemento selecionado, para que seja possível a partir desta chave carregar na tela todos os dados.



Como podemos ver, para a invocação de outros objetos web, poderá ser usado tanto o parâmetro e função Link como a invocação direta do objeto.

Embora neste exemplo possamos usar qualquer uma das duas opções, existem diferenças entre elas.
Por exemplo, se o objeto que queremos chamar for um procedimento, só poderemos fazer isso por invocação direta com o nome do objeto.

Exemplo:

```
Event &ProcedureLink.Click
    Procedure()
EndEvent
```

Por outro lado, se, por exemplo, queremos fazer referência a uma página HTML estática, deveremos utilizar a função Link.

Exemplo:

```
Event Enter
    Link('http://www.genexus.com')
EndEvent
```

Agora voltamos à seção eventos de nosso Web Panel.

Testemos e pratiquemos o uso que acabamos de ver das sub-rotinas, da mesma maneira que as utiliza no Work With.

Desta forma, teremos um código mais limpo e, com possibilidade de em algum momento precisar reutilizar essas sub-rotinas em algum outro evento dentro do mesmo objeto.

Criaremos primeiro a sub-rotina na qual salvaremos os dados dos filtros.

Para isto utilizaremos o comando ‘Sub’ e colocaremos o mesmo nome que lhe atribuiu o Work With ‘SaveGridState’

Agora, inseriremos o código criado anteriormente que programamos no evento &update.click, onde atribuímos os métodos set à variável webSession, passando por parâmetro como chave e valor, cada variável.

Em seguida, fazemos o mesmo, mas com a sub-rotina ‘LoadGridState’. Onde recuperaremos os valores passando uma chave, e esse valor o atribuímos a cada variável de filtro. Tínhamos isso localizado até o momento no evento Start.

Agora só precisamos chamar essas sub-rotinas a partir dos eventos correspondentes.

No caso da sub-rotina ‘SaveGridState’, a chamaremos no evento &update.Click, pelos motivos explicados no vídeo anterior. Embora vimos que, se fizermos isso a partir do evento Refresh, como o Work With gera automaticamente, também atenderá aos nossos requisitos.

E agora, onde devemos fazer a chamada para a sub-rotina ‘LoadGridState’?

Como já analisamos, será no evento Start.

Dessa maneira, criamos uma mistura entre como havíamos implementado à nossa maneira no começo, e como implementa automaticamente o Work With.

Inclusive, se quiséssemos, também poderíamos implementá-lo com um objeto SDT e uma variável deste tipo de dados, da mesma maneira que faz o Pattern.
Não vamos fazer isso neste exemplo, mas seria uma boa prática que, após este vídeo, você tente fazer isso sozinho.

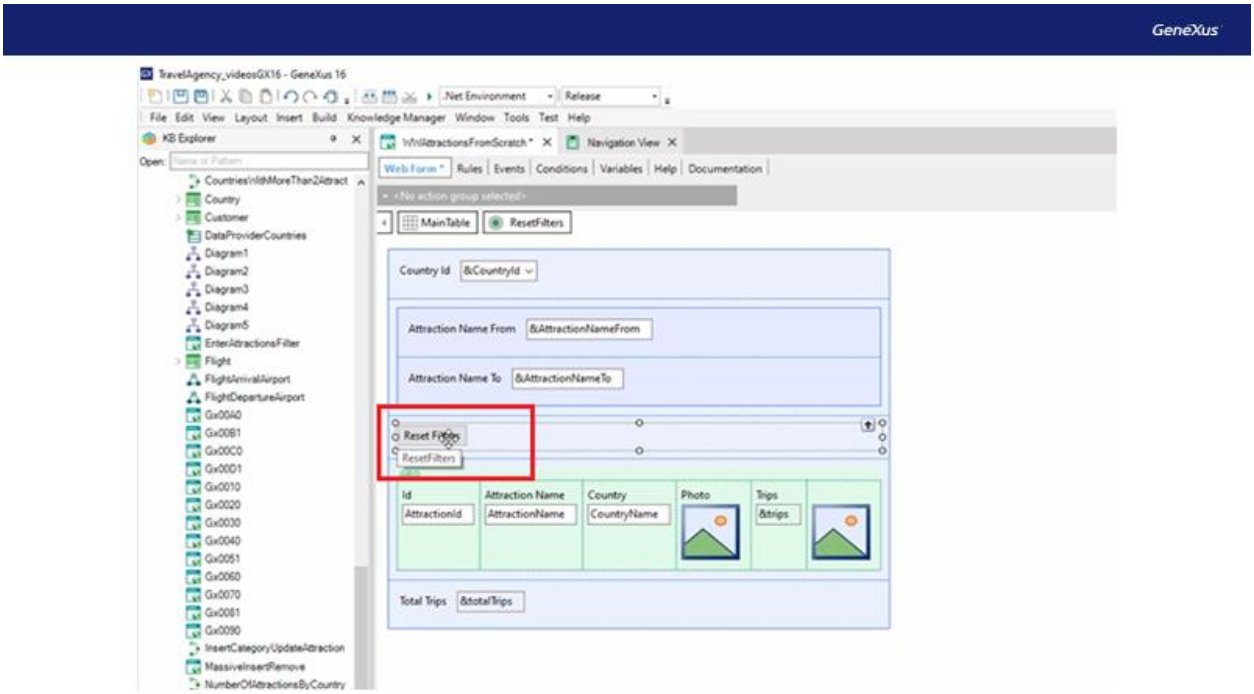
Que os filtros permaneçam entre execuções de telas, pode ser vantajoso quando buscamos esta funcionalidade. Mas pode ser indesejado quando não queremos que se mantenham. Já que deverão ser eliminados um a um os filtros que tenhamos inserido.

Como poderíamos fazer para limpar mais facilmente estes filtros?

Uma das opções seria adicionar em nosso Web Panel um botão que implemente essa funcionalidade.
Basicamente, precisaremos que este botão cumpra três funções:

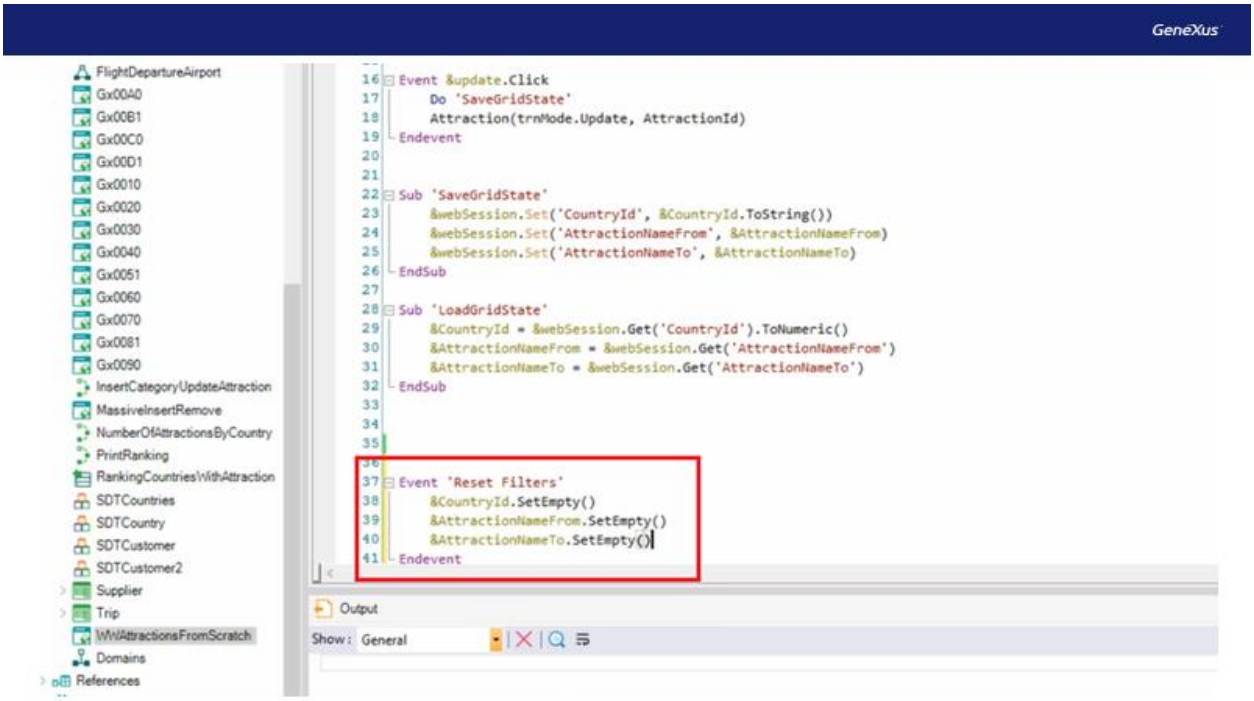
- Esvaziar as variáveis que utilizamos para os filtros.
- Esvaziar a variável do tipo Web Session.
- Atualizar o grid

Para isto arrastaremos um controle do tipo Button para o Web Panel e colocaremos como nome do evento “Reset Filters”



Em seguida, clicamos duas vezes no botão e nos levará para programar o evento do mesmo.

Como dissemos, a primeira das funcionalidades de que precisamos será que as variáveis utilizadas para os filtros fiquem vazias.
Para isto, aplicaremos o método SetEmpty() a cada variável. Dessa maneira, “esvaziamos” cada uma delas.



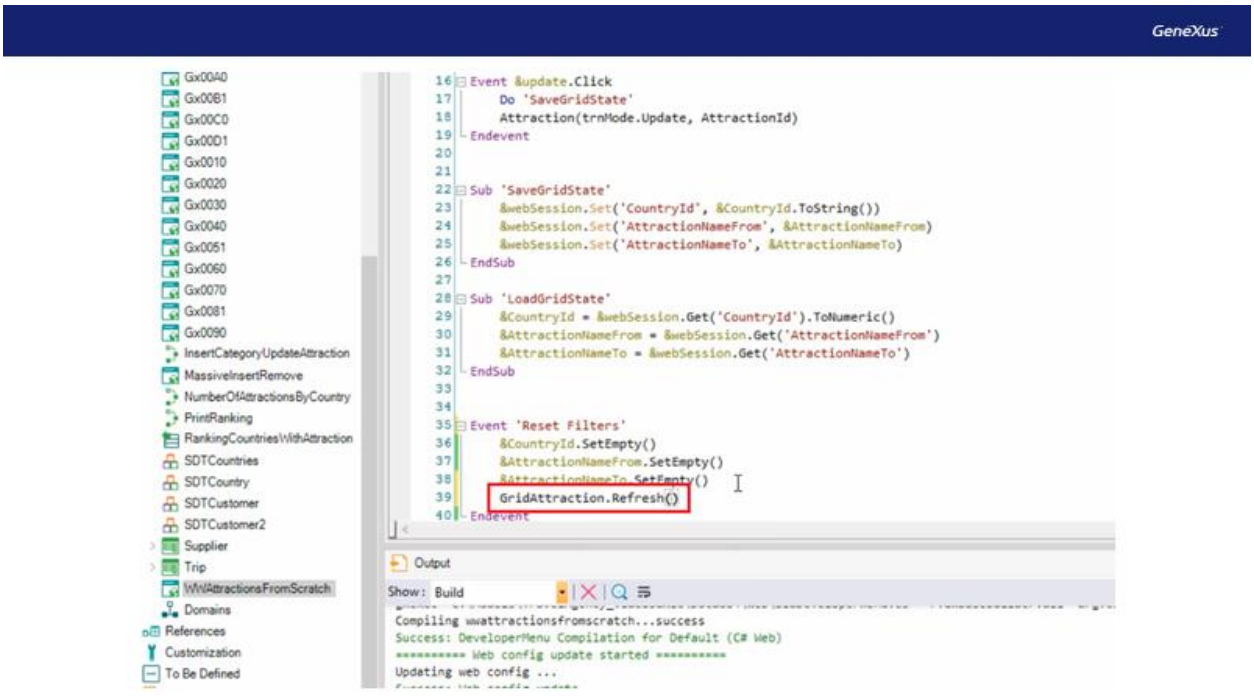
Tentemos executar agora.

Inserimos, por exemplo, o País China e filtraremos as atrações de A a J.

Agora clicamos no botão “Reset Filters”.

Vemos que os filtros ficam vazios, mas o grid não é atualizado.
Isto ocorre porque não foi dada a ordem para o grid atualizar-se. Como fazemos isto?

Deveremos adicionar dentro do evento do botão, o método Refresh() ao grid que queremos que seja atualizado, neste caso, de nome GridAttraction



Agora sim, vamos executar novamente e tentar inserir valores nos filtros e, em seguida, clicar no botão Reset Filters. Vemos que agora funciona corretamente e atualiza o grid como queríamos.

Desta maneira, vimos como o Work With do Pattern implementa a funcionalidade para salvar os valores das variáveis que impactam no grid. Como são, a ordem aplicada ao grid, os valores dos campos editáveis de filtro e o número da página correspondente ao grid.
E comparamos esta solução com a que foi feita à nossa maneira, que propusemos no primeiro vídeo, explicando as diferenças entre elas.
Finalmente, fizemos modificações em nosso WebPanel utilizando sub-rotinas, para explicar seu funcionamento e entender seu uso, assim como faz o WorkWith.

Convidamos vocês a entrar em nossa Wiki para se aprofundar nos temas vistos:
https://wiki.genexus.com/