

Eventos



Behavior: events

Cecilia Fernández | GeneXus Training

Orders & Searches & Conditions

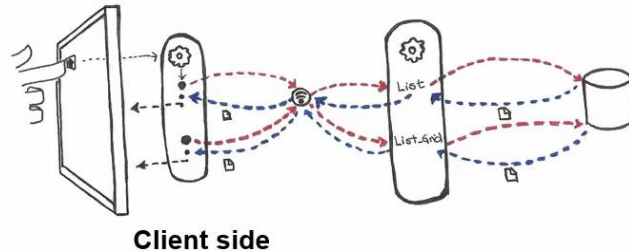
Invocations

Events

Caching

Neste vídeo começaremos a estudar os eventos que podemos definir a nível dos objetos Smart Devices, no contexto de uma aplicação online a qual requer conectividade.

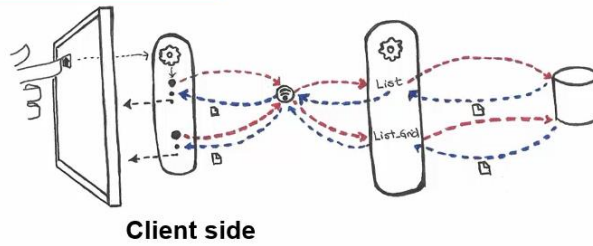
É mencionado no entanto, os pontos em que uma aplicação offline difere do que foi estudado.



- **ClientStart**
- WW predefined events
- User Events
- **Back event**
- Control events

Teremos eventos que executam no cliente, eventos do sistema, como o novo ClientStart ou o Back.

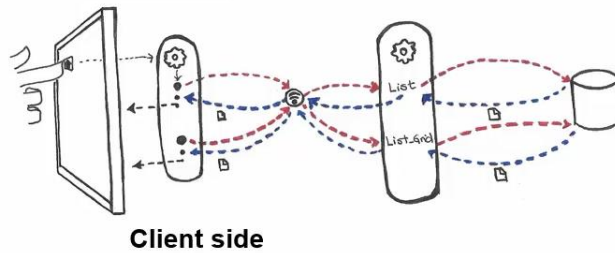
Events



- **ClientStart**
- WW predefined events
- User Events
- **Back event**
- Control events

E de outros: os eventos de usuario e associados à controles

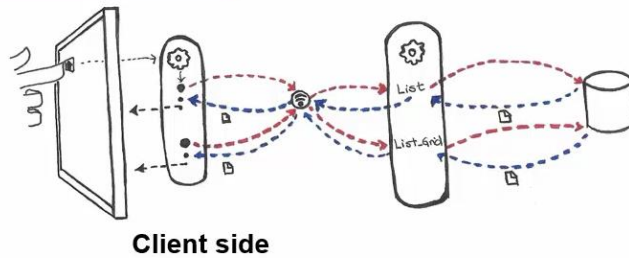
Events



- **ClientStart**
- WW predefined events
- **User Events**
- **Back event**
- **Control events**

entre os que encontram-se predefinidos pelo pattern work with.

Events

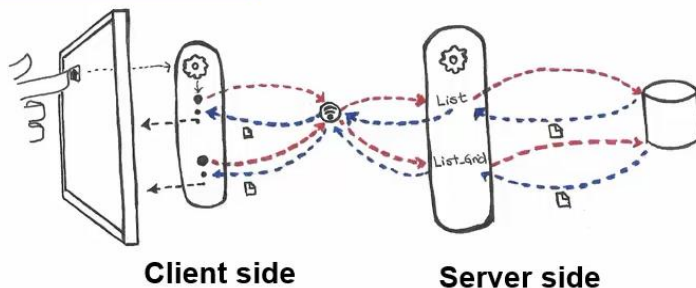


- **ClientStart**
- WW predefined events
- **User Events**
- **Back event**
- **Control events**



Teremos os eventos do sistema já conhecidos: Start, Refresh e Load, os quais executam no servidor.

Events



- **ClientStart**
- WW predefined events
- User Events
- **Back event**
- **Control events**

- Start
- Refresh
- Load

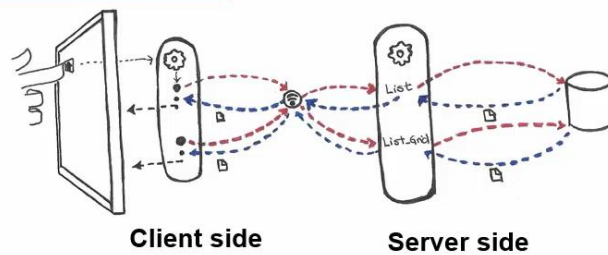
If
Online

Sempre e quando trata-se de um objeto o qual requer conexão online.

Logo veremos o que acontece com estes eventos no caso de uma arquitetura offline, mas podemos adiantar que executam no cliente.

Quando trata-se do objeto main da aplicação, também teremos os novos eventos.

Events



{Flip/Split/Slide/Cascade/Tabs}.Start

- **ClientStart**
- WW predefined events
- User Events
- **Back event**
- Control events

- Start
- Refresh
- Load

If
Online

Flip.Start

Split.Start

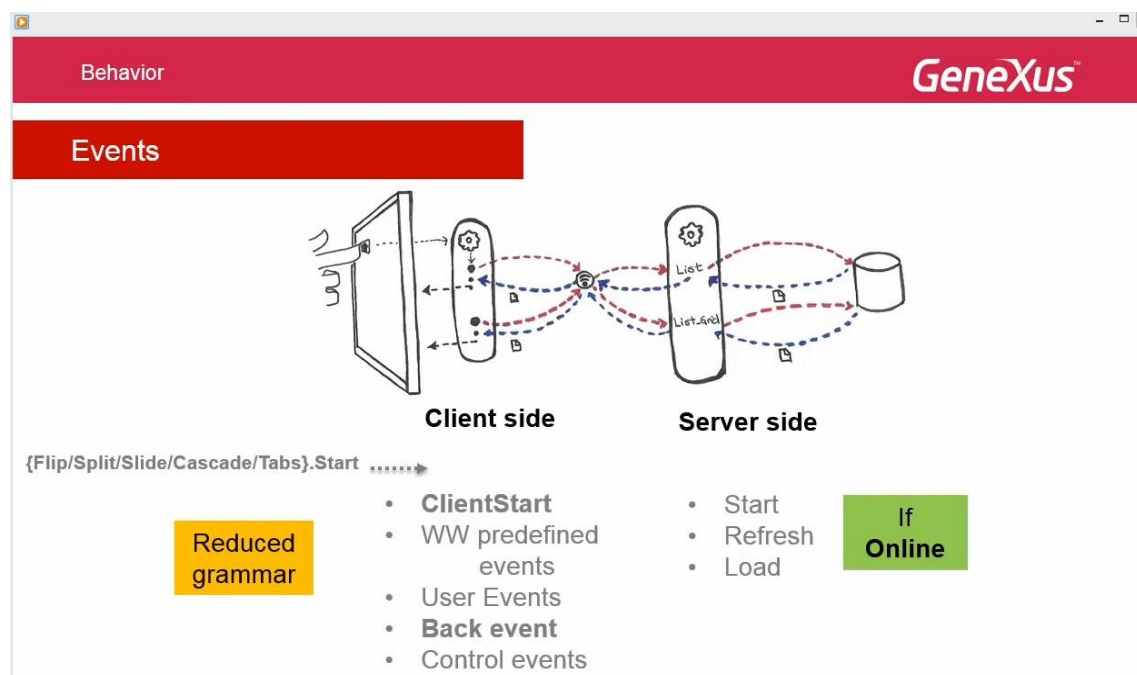
Slide.Start

Cascade.Start

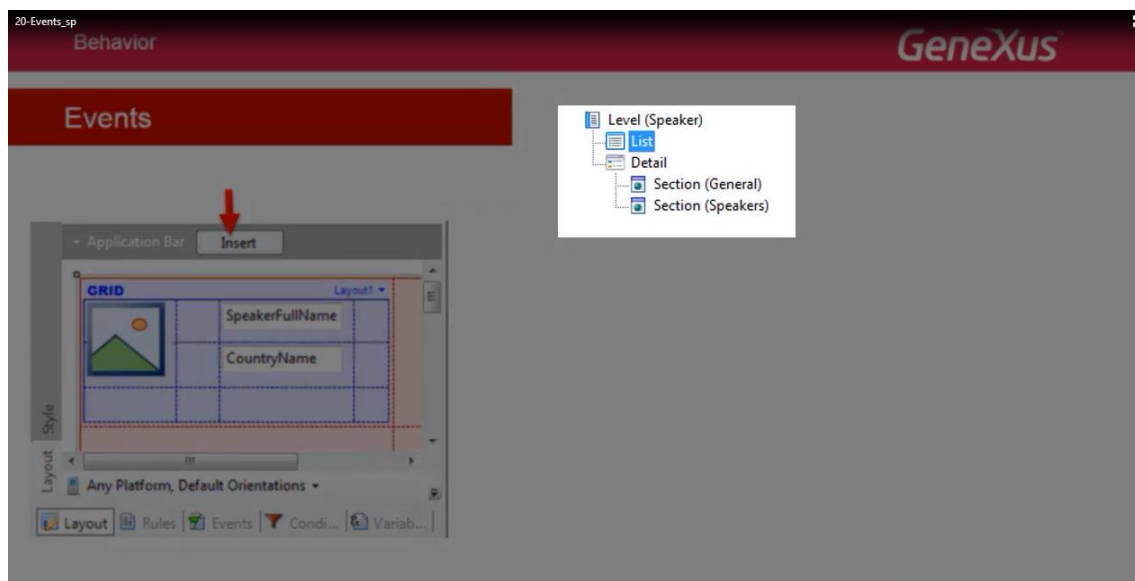
E Tabs.Start

O qual vimos quando estudamos o navigation style da aplicação. Somente são válidos para o objeto principal, seja este um Dashboard ou não.

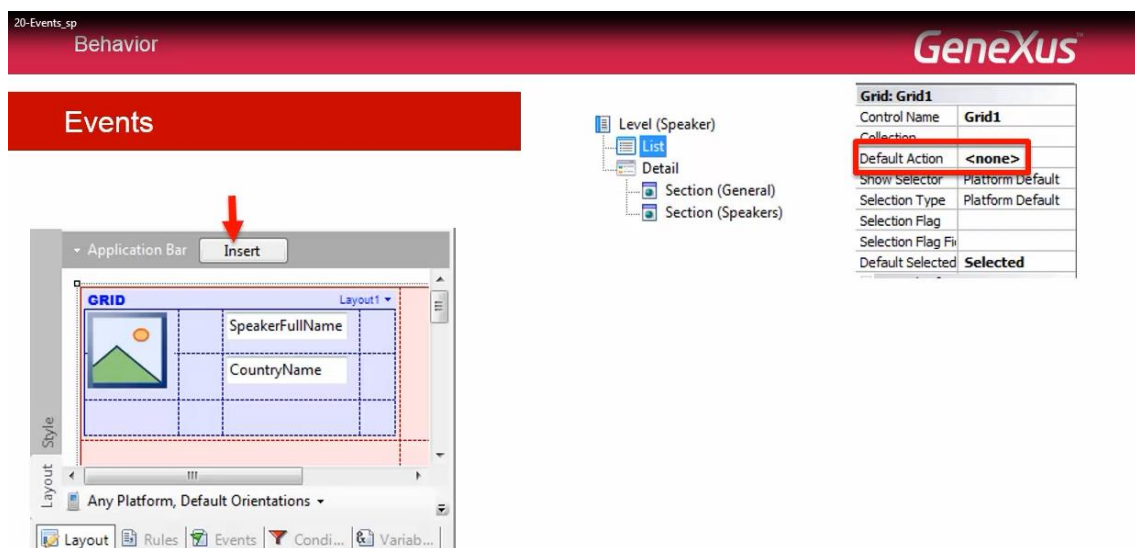
Os eventos executados no cliente, tem uma gramática reduzida sobre eventos no servidor, gramática que veremos mais adiante em outro vídeo.



No exemplo, o evento Insert no List.



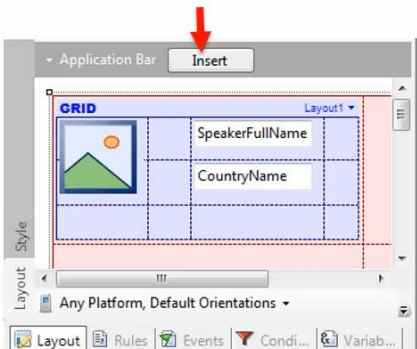
É um evento criado pelo pattern.



Também por conta do pattern, associa-se à propriedade Default Action, o valor: default.

20-Events_sp Behavior **GeneXus**

Events



Level (Speaker)
List
Detail
Section (General)
Section (Speakers)

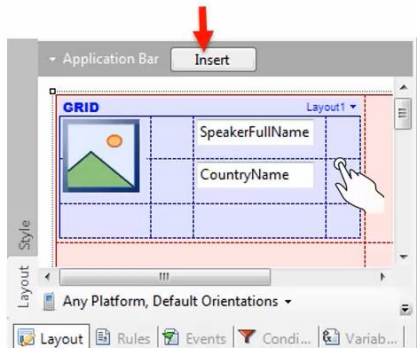
Grid: Grid1	
Control Name	Grid1
Collection	
Default Action	<none>
Show Selector	Platform Default
Selection Type	Platform Default
Selection Flag	
Selection Flag Fi	
Default Selected	Selected

<default>

O qual indica o que não vemos em um evento associado, chamado Detail em modo View quando o usuário fazia o TAP,

20-Events_sp Behavior **GeneXus**

Events



Level (Speaker)
List
Detail
Section (General)
Section (Speakers)

Grid: Grid1	
Control Name	Grid1
Collection	
Default Action	<none>
Show Selector	Platform Default
Selection Type	Platform Default
Selection Flag	
Selection Flag Fi	
Default Selected	Selected

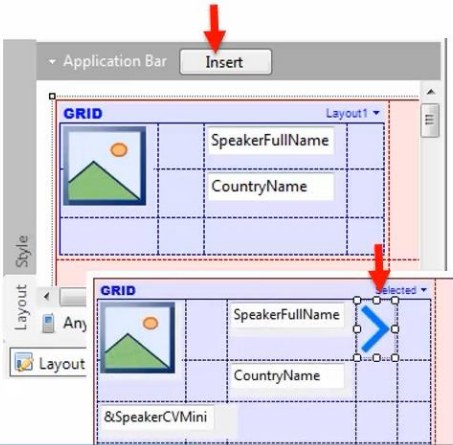
<default>

sobre uma linha do grid.

Aqui mudamos este comportamento pois programamos outros através de um dos eventos de controles.

20-Events_sp Behavior **GeneXus**

Events



Level (Speaker)

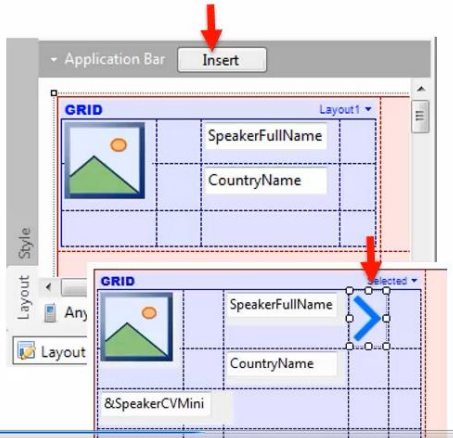
- List
- Detail
 - Section (General)
 - Section (Speakers)

Grid: Grid1	
Control Name	Grid1
Collection	
Default Action	<none>
Show Selector	Platform Default
Selection Type	Platform Default
Selection Flag	
Selection Flag Fi	
Default Selected	Selected

O qual tem a ver com a lógica "touch screen".

20-Events_sp Behavior **GeneXus**

Events



Level (Speaker)

- List
- Detail
 - Section (General)
 - Section (Speakers)

Grid: Grid1	
Control Name	Grid1
Collection	
Default Action	<none>
Show Selector	Platform Default
Selection Type	Platform Default
Selection Flag	
Selection Flag Fi	
Default Selected	Selected

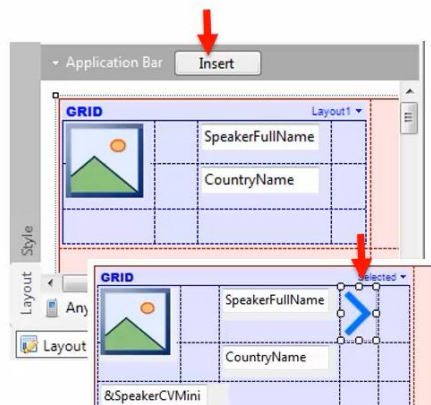
```

1 Event 'Insert'
2   WorkWithDevicesSpeaker.Speaker.Detail.Insert ()
3 EndEvent
4
5 Event arrow.Tap
6   WorkWithDevicesSpeaker.Speaker.Detail (SpeakerId)
7 EndEvent
8
9 Event Load
10  Table1.Class = "TableColoredWhite"
11  SpeakerFullName.Class = "AttributeFontColorViolet"
12  if SpeakerCVMini.Length() > 145
13    &SpeakerCVMini = substr( SpeakerCVMini, 1, 145)
14  else
15    &SpeakerCVMini = SpeakerCVMini
16  endif
17 EndEvent
  
```

O evento Tap sobre a imagem de nome arrow.

20-Events_sp
Behavior
GeneXus

Events



Grid: Grid1	
Control Name	Grid1
Collection	
Default Action	<none>
Show Selector	Platform Default
Selection Type	Platform Default
Selection Flag	
Selection Flag Fv	
Default Selected	Selected

```

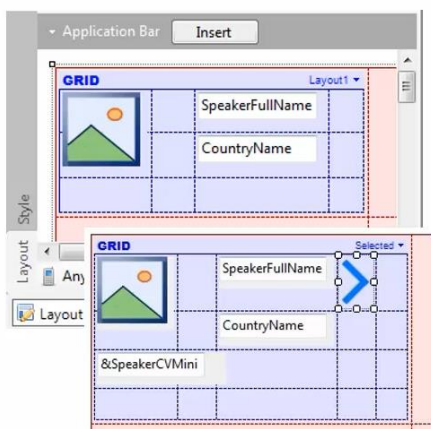
1 Event 'Insert'
2   WorkWithDevicesSpeaker.Speaker.Detail.Insert ()
3 EndEvent
4
5 Event arrow.Tap
6   WorkWithDevicesSpeaker.Speaker.Detail (SpeakerId)
7 EndEvent
8
9 Event Load
10  Table1.Class = "TableColoredWhite"
11  SpeakerFullName.Class = "AttributeFontColorViolet"
12  if SpeakerCVMMini.Length() > 145
13    &SpeakerCVMMini = substr( SpeakerCVMMini, 1, 145)
14  else
15    &SpeakerCVMMini = SpeakerCVMMini
16  endif
17 EndEvent
          
```


Todos estes eventos são do cliente, embora invoquem objetos que devam chamar serviços – data providers – do servidor para devolver os dados ou gravá-los.

Na mudança, o evento Load para carregar as linhas do grid.

Behavior
GeneXus

Events





If Online

```

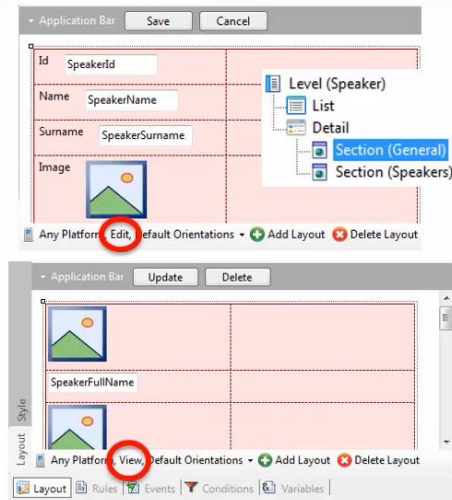
1 Event 'Insert'
2   WorkWithDevicesSpeaker.Speaker.Detail.Insert ()
3 EndEvent
4
5 Event arrow.Tap
6   WorkWithDevicesSpeaker.Speaker.Detail (SpeakerId)
7 EndEvent
8
9 Event Load
10  Table1.Class = "TableColoredWhite"
11  SpeakerFullName.Class = "AttributeFontColorViolet"
12  if SpeakerCVMMini.Length() > 145
13    &SpeakerCVMMini = substr( SpeakerCVMMini, 1, 145)
14  else
15    &SpeakerCVMMini = SpeakerCVMMini
16  endif
17 EndEvent
          
```

Será executado no servidor se o objeto executa-se online.

Behavior

GeneXus

Events

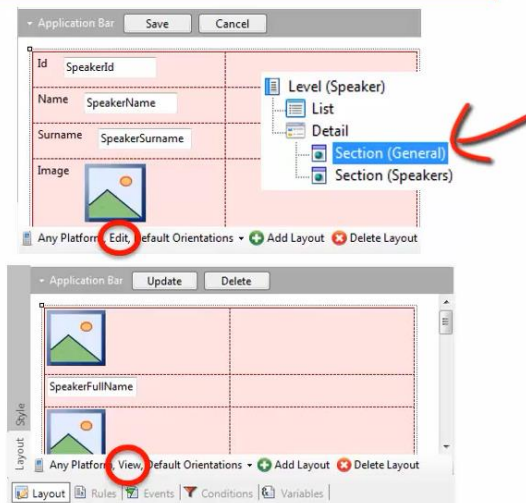


Vemos aqui os layouts criados pelo pattern para a section: General.

Behavior

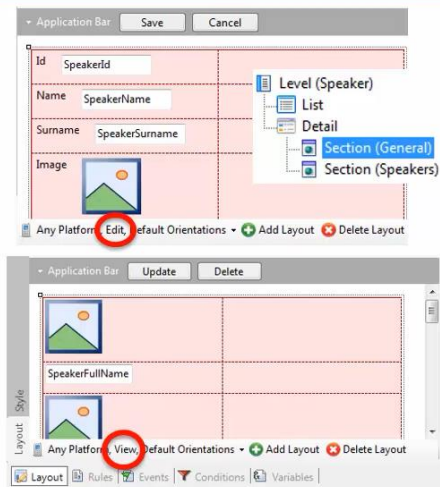
GeneXus

Events



E os eventos Default.

Events



```

Event 'Save'
  Composite
    SDActions.Save ()
    return
  EndComposite
EndEvent

Event 'Cancel'
  SDActions.Cancel ()
EndEvent

Event 'Update'
  WorkWithDevicesSpeaker.Speaker.Detail.Update (SpeakerId)
EndEvent

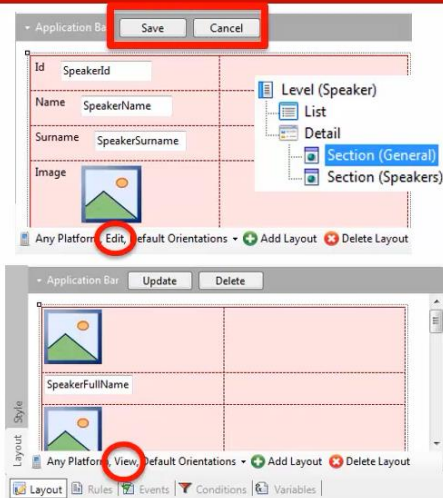
Event 'Delete'
  Composite
    WorkWithDevicesSpeaker.Speaker.Detail.Delete (SpeakerId)
    return
  EndComposite
EndEvent

```

São quatro.

Dois aplicam o modo Edit.

Events



```

Event 'Save'
  Composite
    SDActions.Save ()
    return
  EndComposite
EndEvent

Event 'Cancel'
  SDActions.Cancel ()
EndEvent

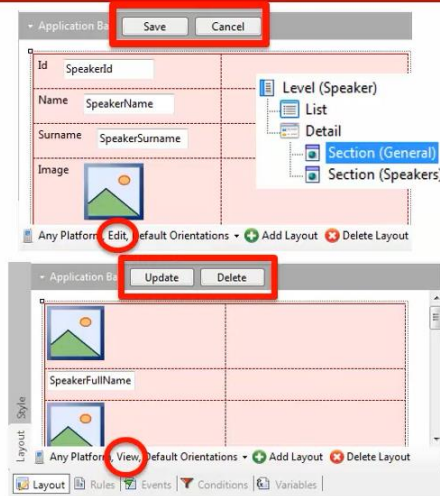
Event 'Update'
  WorkWithDevicesSpeaker.Speaker.Detail.Update (SpeakerId)
EndEvent

Event 'Delete'
  Composite
    WorkWithDevicesSpeaker.Speaker.Detail.Delete (SpeakerId)
    return
  EndComposite
EndEvent

```

Os outros aplicam o modo View..

Events



```

Event 'Save'
  Composite
    SDActions.Save ()
    return
  EndComposite
EndEvent

Event 'Cancel'
  SDActions.Cancel ()
EndEvent

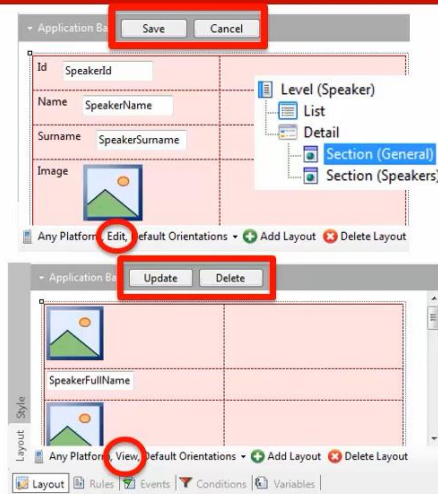
Event 'Update'
  WorkWithDevicesSpeaker.Speaker.Detail.Update (SpeakerId)
EndEvent

Event 'Delete'
  Composite
    WorkWithDevicesSpeaker.Speaker.Detail.Delete (SpeakerId)
    return
  EndComposite
EndEvent

```

Estes também serão eventos do cliente.

Events



```

Event 'Save'
  Composite
    SDActions.Save ()
    return
  EndComposite
EndEvent

Event 'Cancel'
  SDActions.Cancel ()
EndEvent

Event 'Update'
  WorkWithDevicesSpeaker.Speaker.Detail.Update (SpeakerId)
EndEvent

Event 'Delete'
  Composite
    WorkWithDevicesSpeaker.Speaker.Detail.Delete (SpeakerId)
    return
  EndComposite
EndEvent

```

Note o comando Composit.

Events



```

Event 'Save'
  Composite
    SDActions.Save ()
    return
  EndComposite
EndEvent

Event 'Cancel'
  SDActions.Cancel ()
EndEvent

Event 'Update'
  WorkWithDevicesSpeaker.Speaker.Detail.Update (SpeakerId)
EndEvent

Event 'Delete'
  Composite
    WorkWithDevicesSpeaker.Speaker.Detail.Delete (SpeakerId)
    return
  EndComposite
EndEvent

```

O qual estudaremos em breve, no evento Save podemos ver o objeto externo SDActions:

```

Event 'Save'
  Composite
    SDActions.Save ()
    return
  EndComposite
EndEvent

Event 'Cancel'
  SDActions.Cancel ()
EndEvent

Event 'Update'
  WorkWithDevicesSpeaker.Speaker.Detail.Update (SpeakerId)
EndEvent

Event 'Delete'
  Composite
    WorkWithDevicesSpeaker.Speaker.Detail.Delete (SpeakerId)
    return
  EndComposite
EndEvent

```

Encontra-se no folder das APIs sobre o qual falaremos.

O que faz é encapsular a invocação ao Business Component Rest o qual encontra-se no server, para gravar a informação do speaker no banco de dados.

20-Events_sp Behavior GeneXus

Events

```

Event 'Save'
Composite
  SDActions.Save ()
return
EndComposite
EndEvent

Event 'Cancel'
  SDActions.Cancel ()
EndEvent

Event 'Update'
  WorkWithDevicesSpeaker.Speaker.Detail.Update (SpeakerId)
EndEvent

Event 'Delete'
Composite
  WorkWithDevicesSpeaker.Speaker.Detail.Delete (SpeakerId)
return
EndComposite
EndEvent

```

20-Events_sp Behavior GeneXus

```

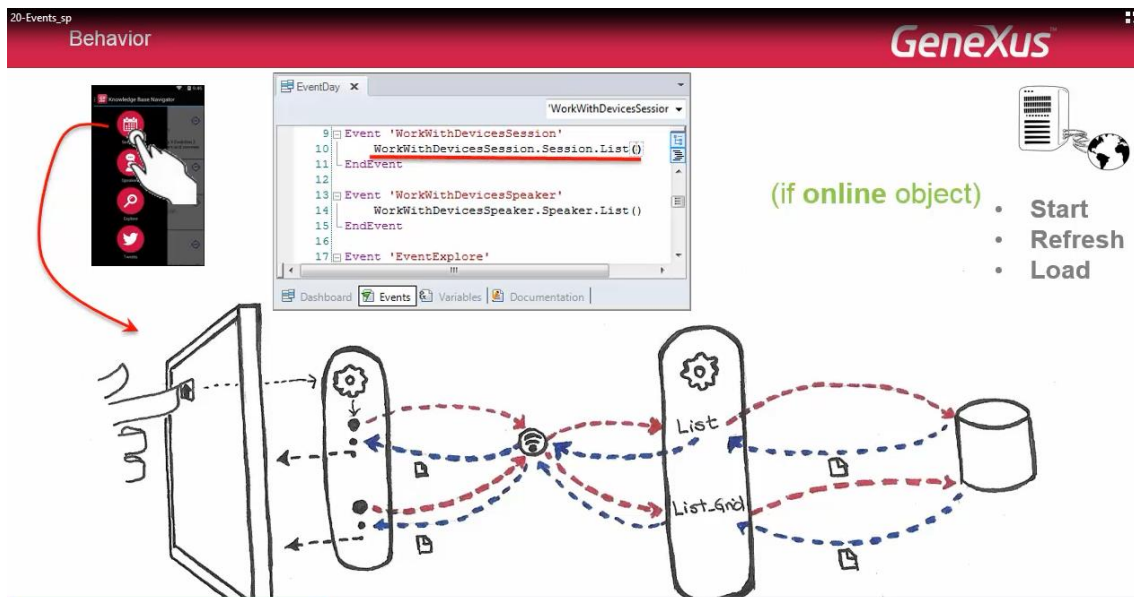
9 Event 'WorkWithDevicesSession'
10   WorkWithDevicesSession.Session.List ()
11 EndEvent
12
13 Event 'WorkWithDevicesSpeaker'
14   WorkWithDevicesSpeaker.Speaker.List ()
15 EndEvent
16
17 Event 'EventExplore'

```

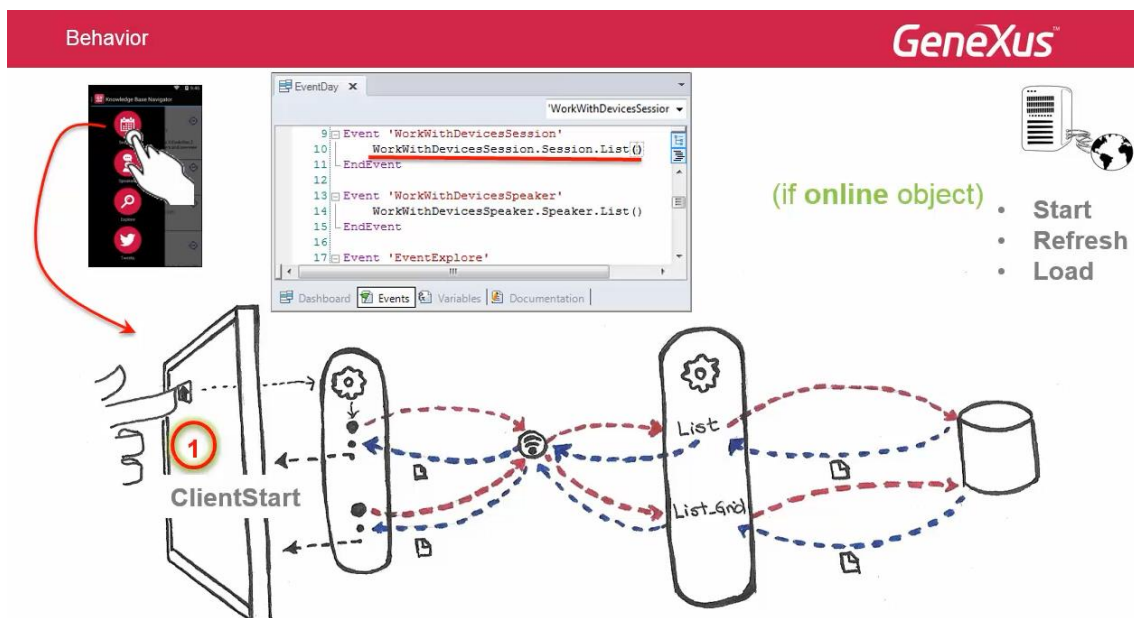
(if online object)

- Start
- Refresh
- Load

Vejamos os eventos do sistema que executam no server no contexto de uma aplicação online.

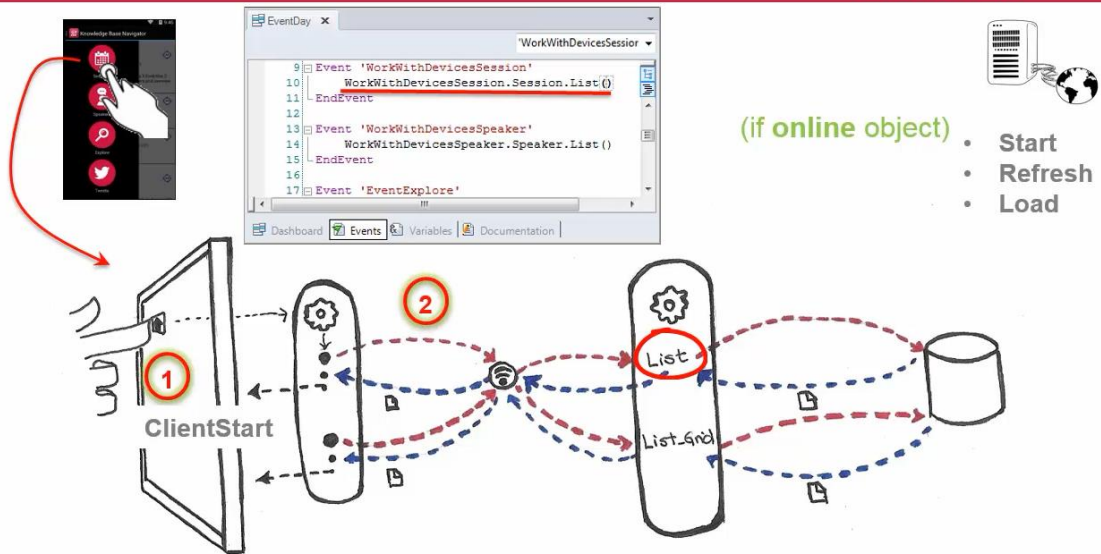


O que acontece quando a partir do Dashboard invocamos a List de Sessions?



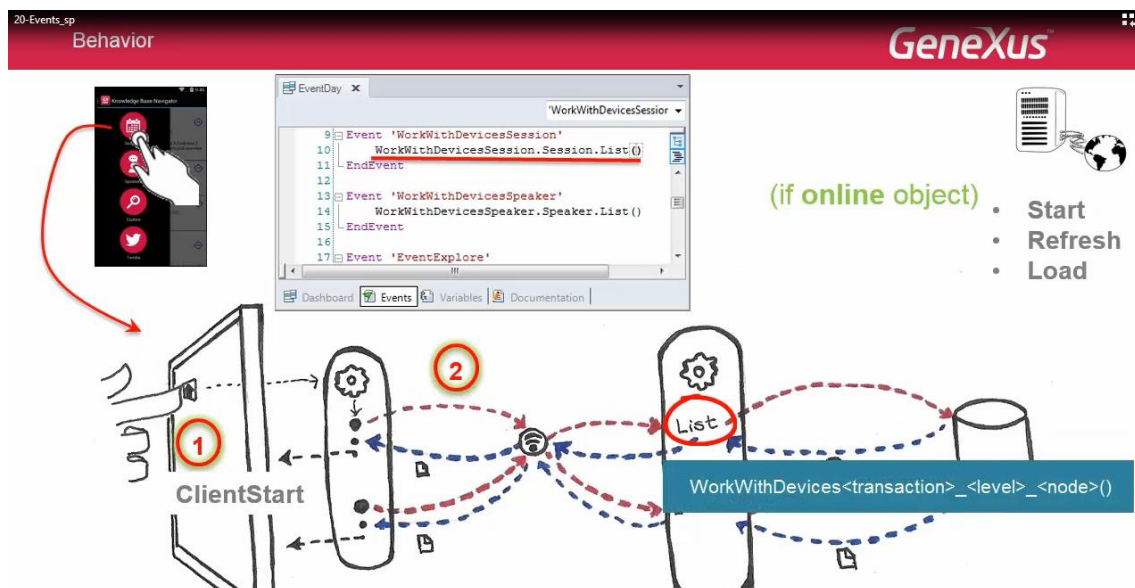
No cliente este List executa o evento ClientStart.

Depois, faz um call externo ao serviço Rest.

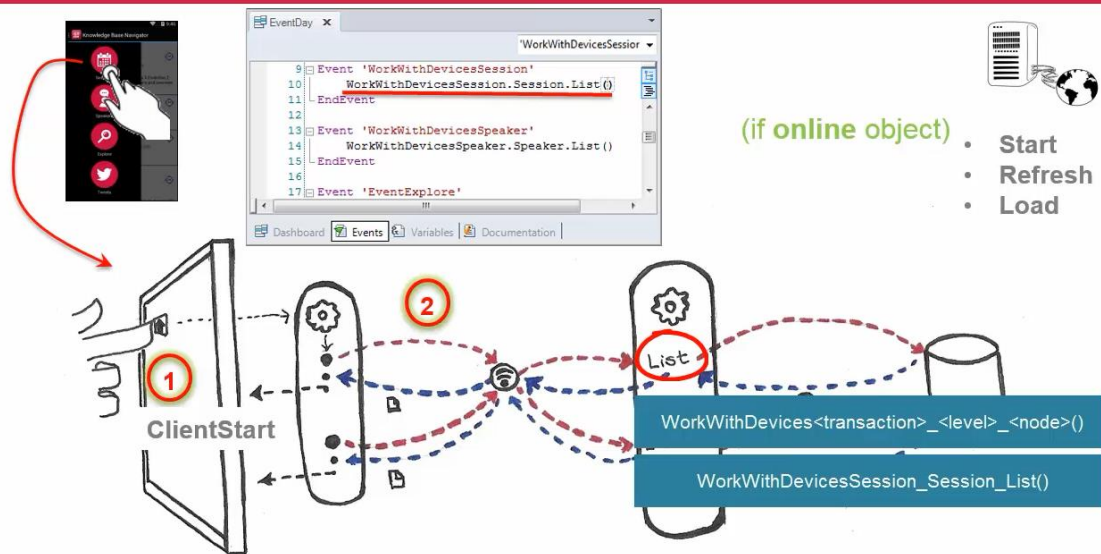


O qual devolverá os dados da parte fixa do panel.

Será um data provider criado automaticamente por GeneXus, não sendo visível no KB mas aparecerá na listagem de navegação com o nome Work With Devices – Nome da transação – Nível do workwith que se trata e o nó que nos encontramos.



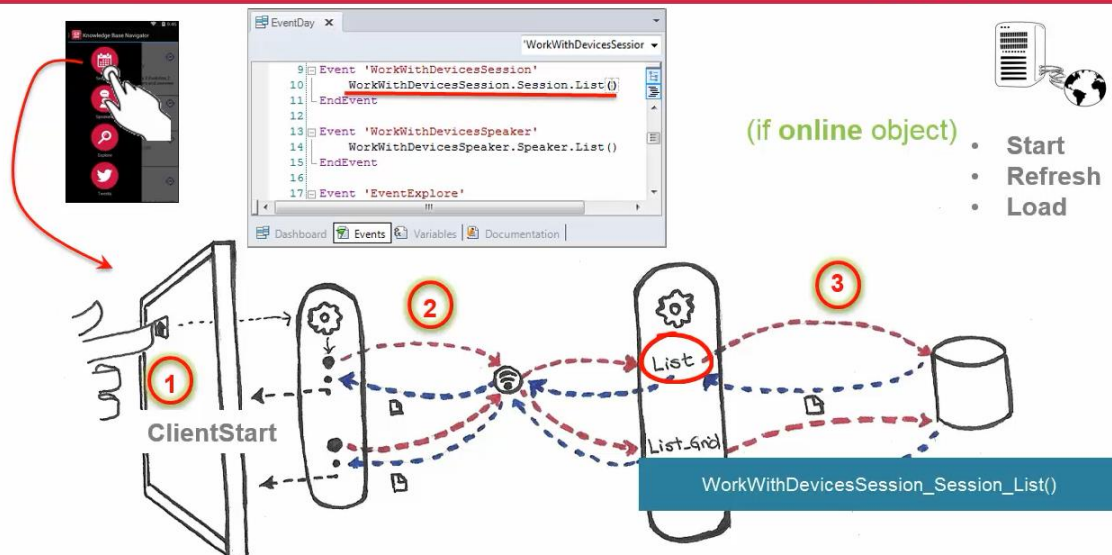
No nosso caso será:



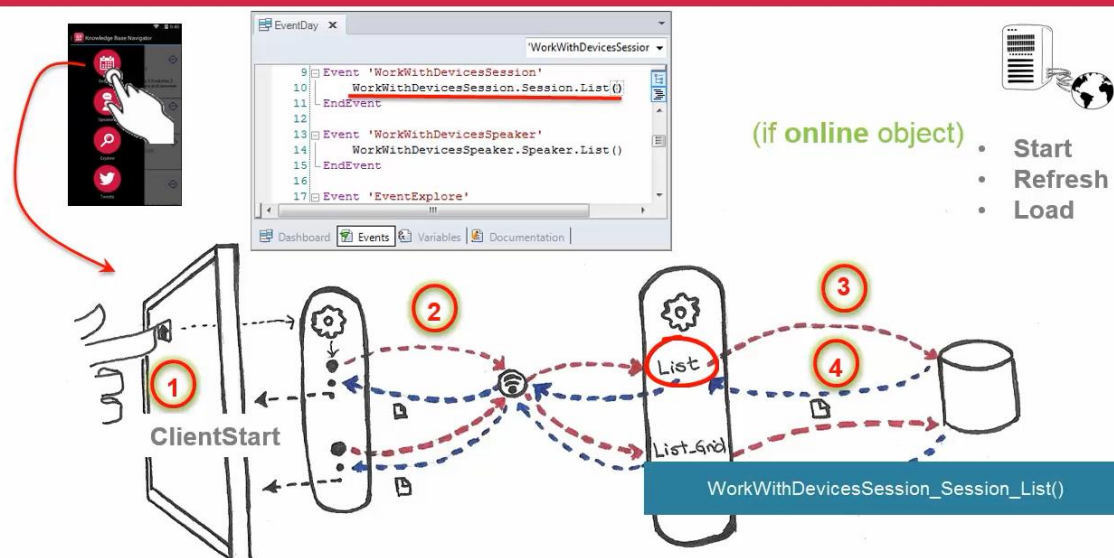
- WorkWithDevicesSession (nome da transação)
- Session (nome do nível)
- List (o nó que estamos chamando)

Este data provider executa no servidor, dentro de sua lógica interna executam os eventos Start e Refresh.

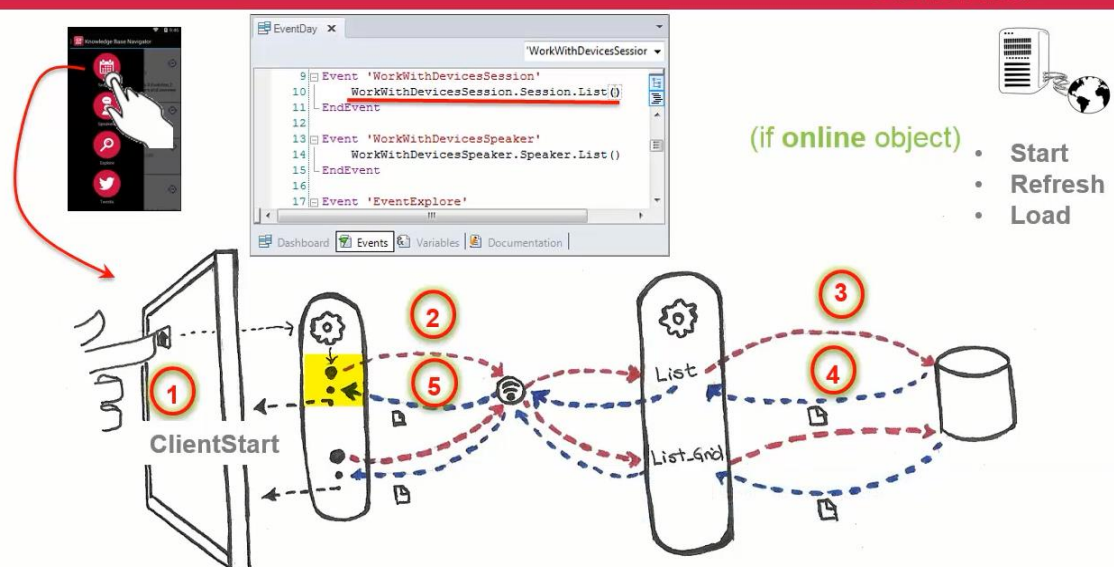
Acessa a base de dados se necessário.



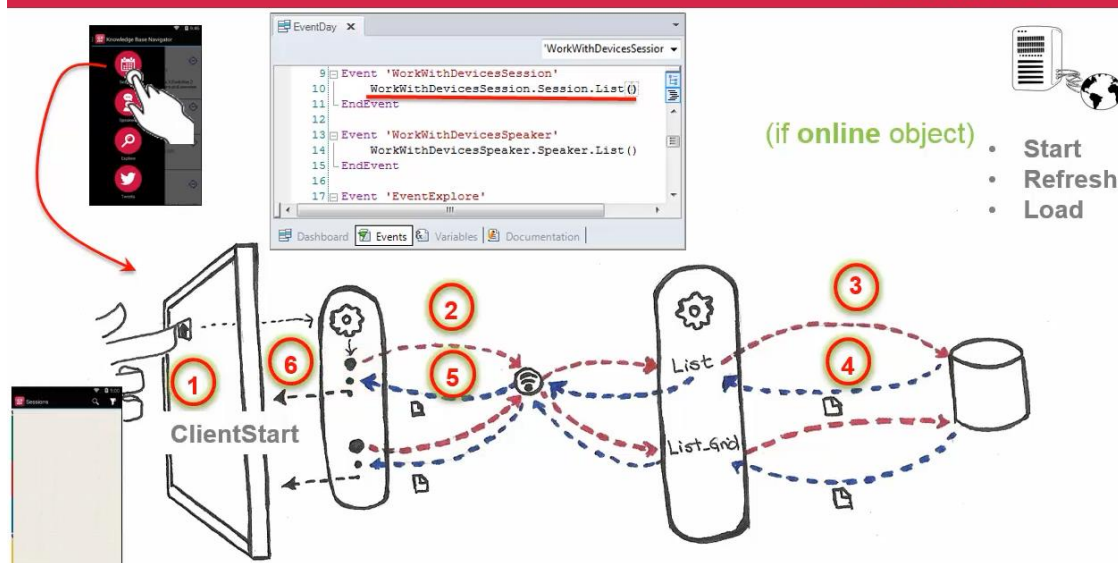
A base de dados encontra os registros e os devolve ao data provider.



Quem devolve a resposta da informação por parte da aplicação que roda no dispositivo é o work with que o chamou:

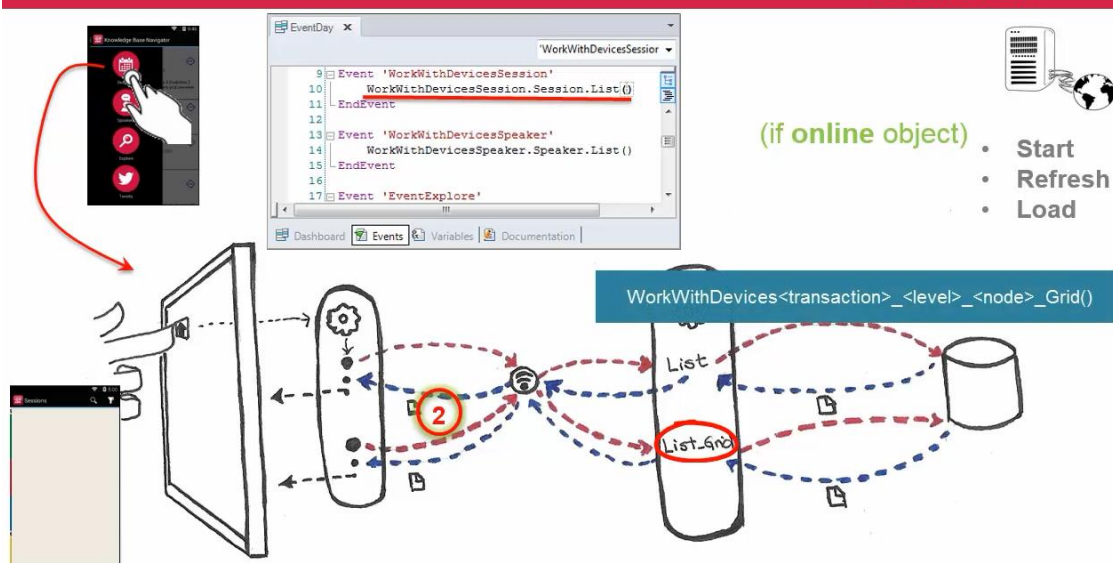


Este work with, busca as imagens que necessita e com a resposta recebida, desenha a interface do usuário correspondente a parte fixa do List do Work With.



Em nosso caso não requer dados.

Logo, repetem os passos anteriores, mas agora chamando o data provider criado automaticamente por GeneXus e transparente para nós, devolvendo os dados do grid.



Seu nome será WorkWithDevices – nome da transação – nível – nó – Grid

No nosso caso, será:



- **Start**
- **Refresh**
- **Load**

```
WorkWithDevicesSession_Session_List_Grid()
```

Behavior

GeneXus™

(if online object)

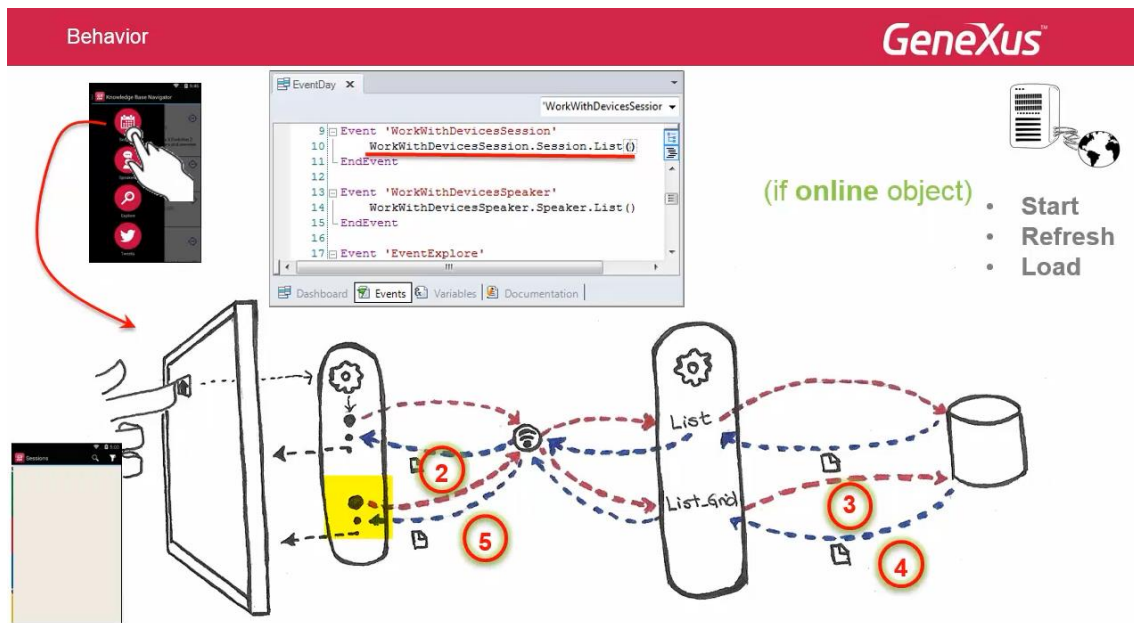
- Start
- Refresh
- Load

WorkWithDevicesSession_Session_List_Grid()

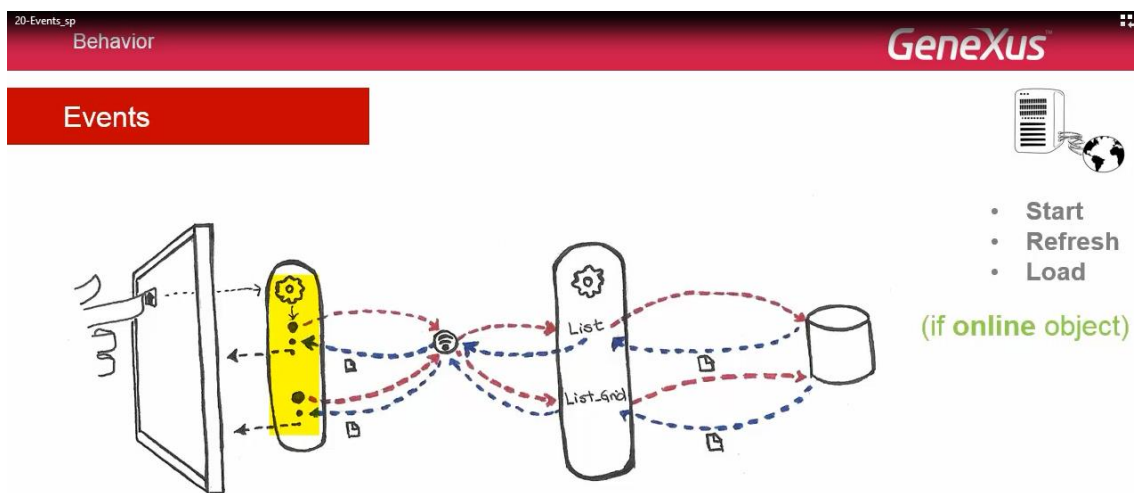
ListGrid

Load

Uma vez que a aplicação no dispositivo recebe a resposta com os dados do grid,



desenha a interface do usuário correspondente ao grid.



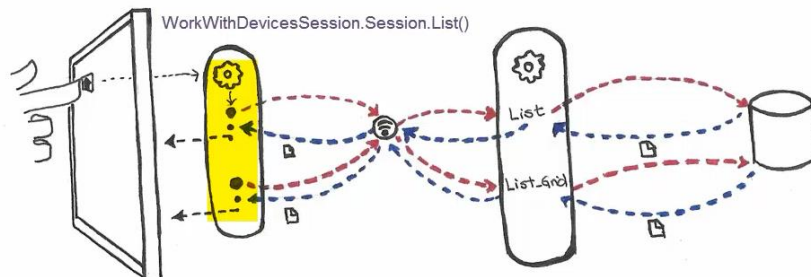
Em definitivo, do lado do cliente, o dispositivo, temos no evento do dashboard, o qual ativamos ao fazer TAP sobre a imagem.

Events



- Start
- Refresh
- Load

(if online object)



Uma chamada ao List do work with de sessions.

O código correspondente é o que roda no cliente, começando a executar esse List.

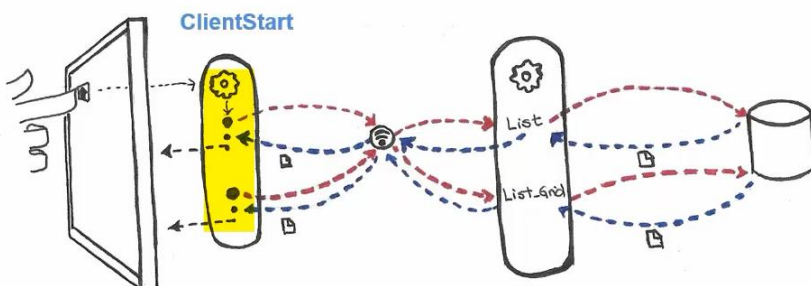
Em sua lógica interna, este work with, executa o evento Client Start

Events



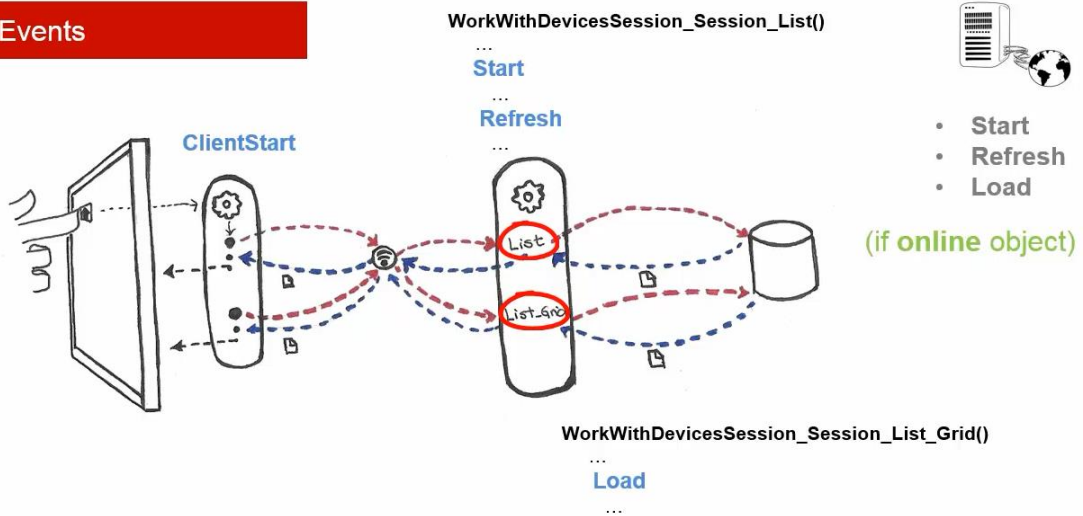
- Start
- Refresh
- Load

(if online object)



Logo, chama o servidor, para que este devolva os dados a serem carregados na parte fixa através de um serviço Rest.

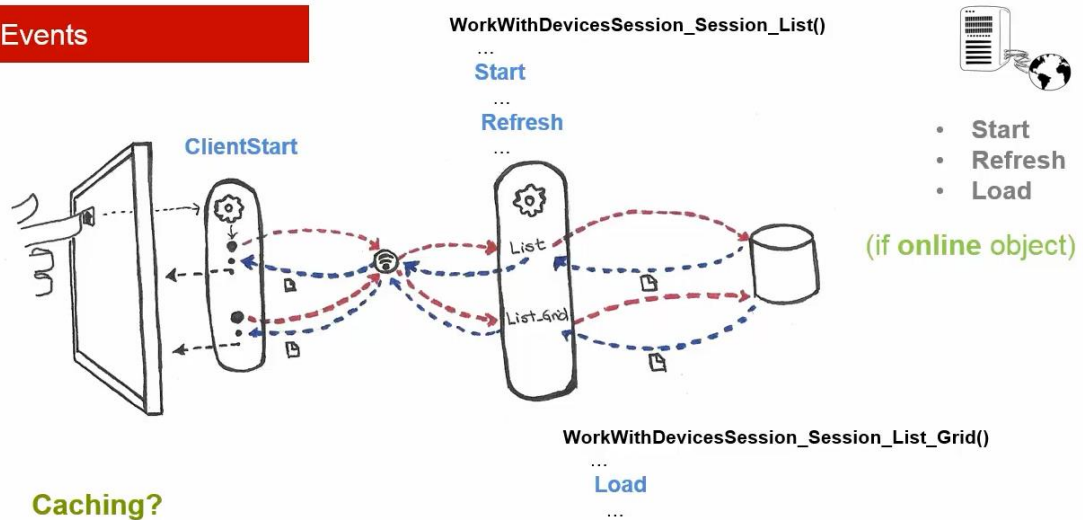
Events



monta a tela: por um lado a parte fixa e pelo outro o grid.

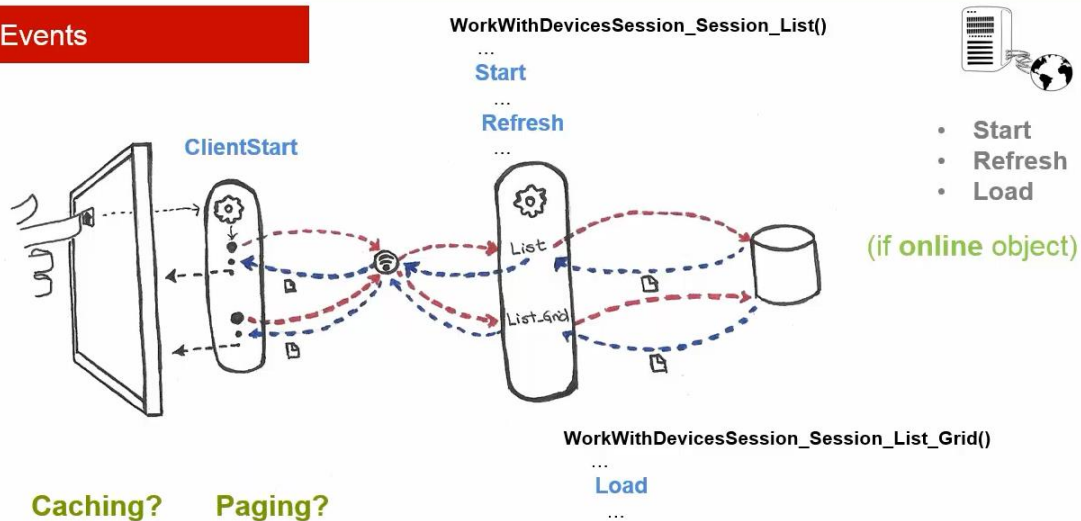
Estamos supondo que não há caching para simplificar a explicação,

Events

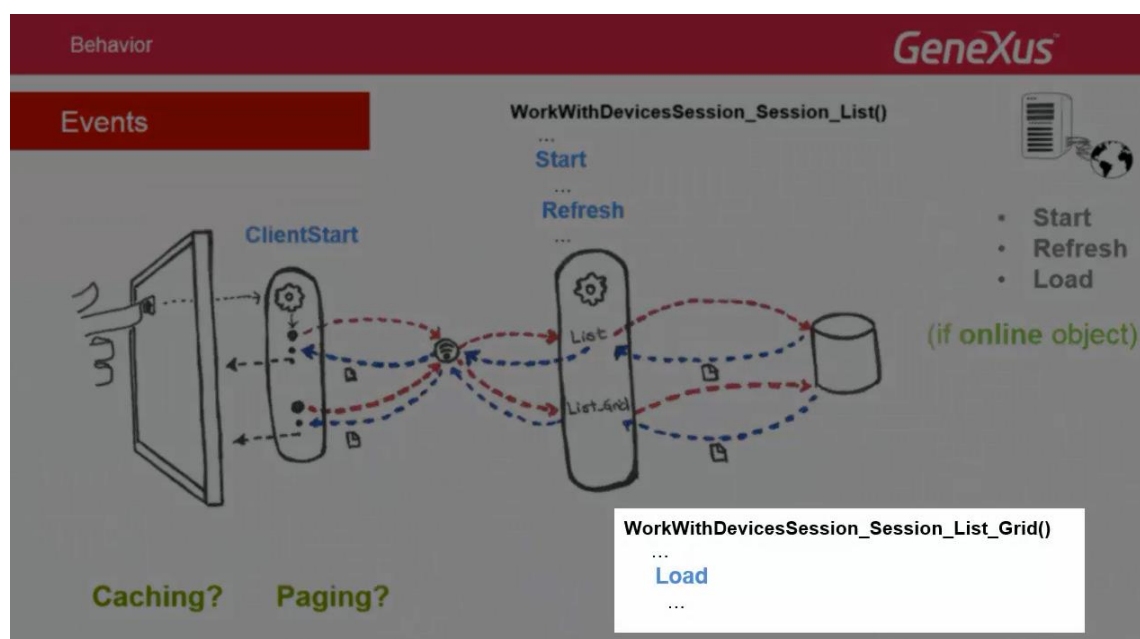


Tampouco estamos com foco no paginado do grid, que também é realizado.

Events

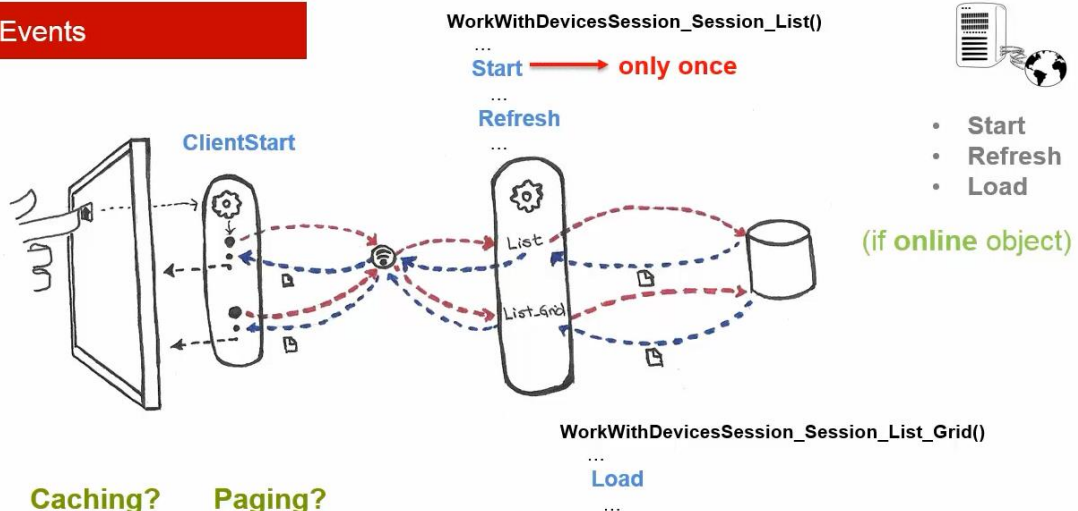


O data provider executa o Load.



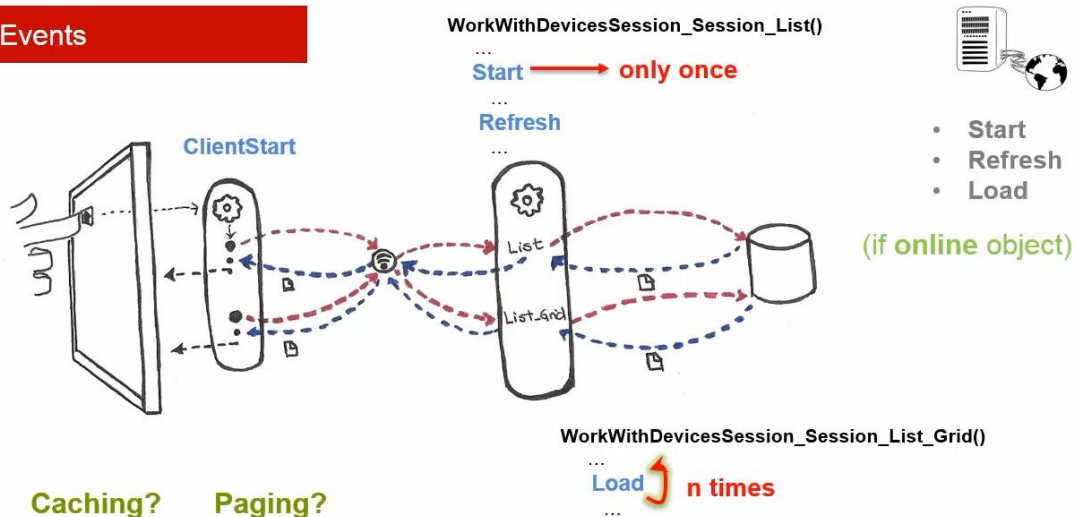
O faz paginando, ou seja, somente devolve X registros por vez.

Events



O evento Start, dispara somente a primeira vez. Não volta a executar, a menos que saia do panel e entre novamente.

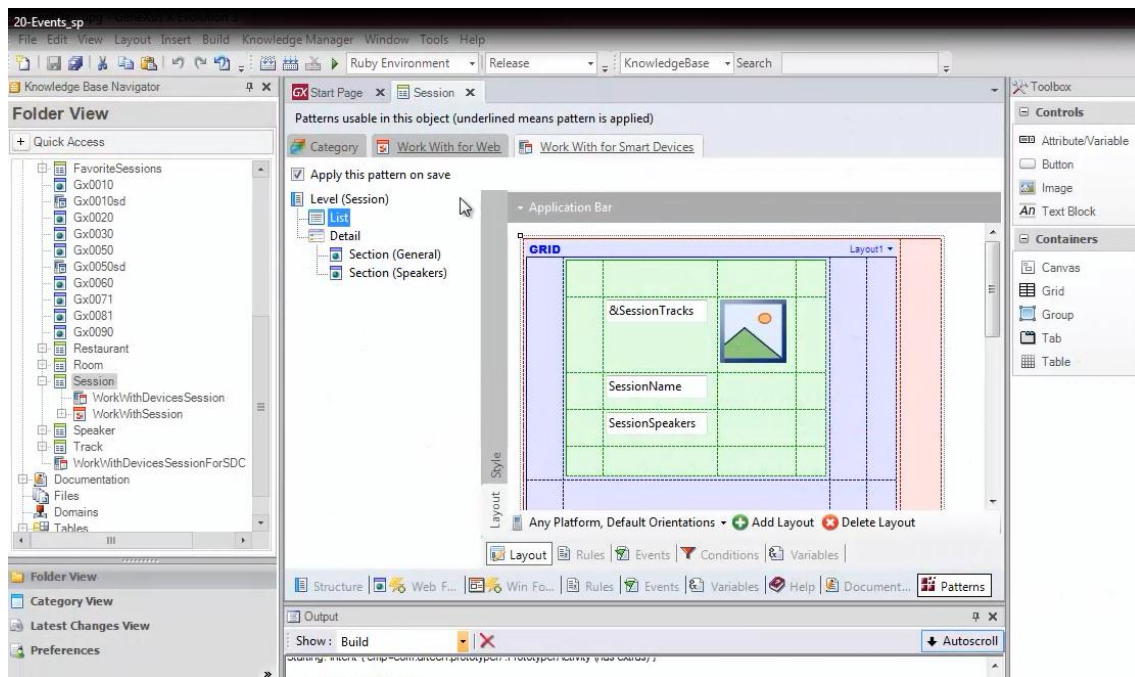
Events



O evento Load no web panel, executa N vezes, se o grid tem tabela base, como é o caso.

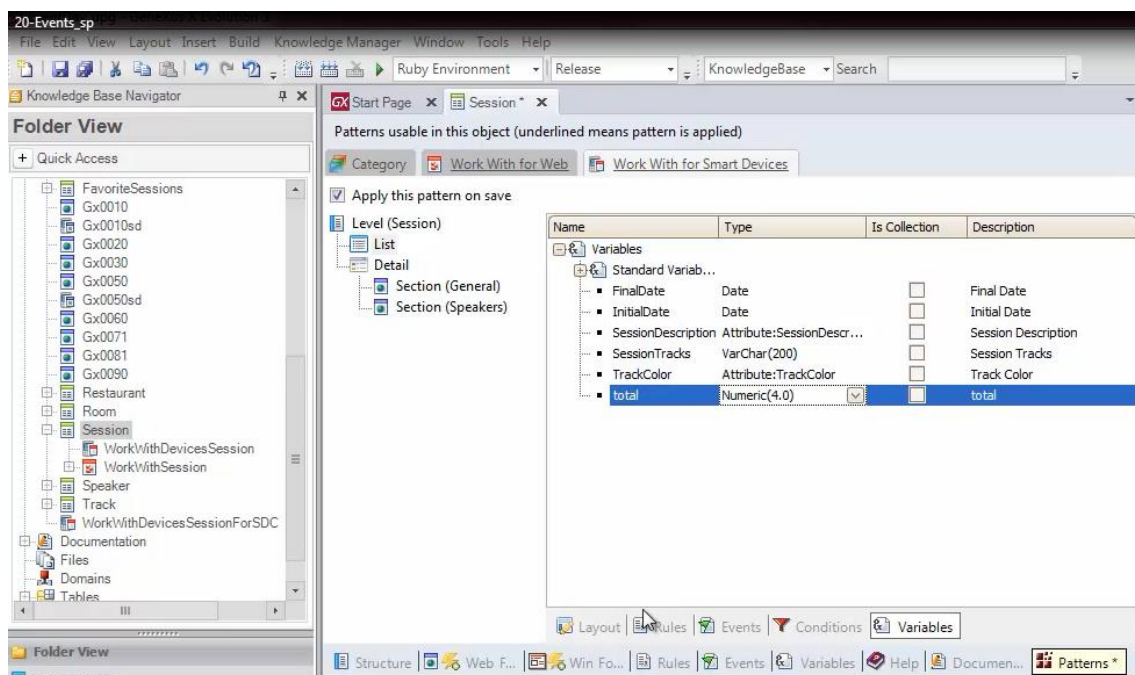
Podemos ver então a diferença entre um panel de Smart Devices (como é o Listo u o Detail) e um web panel.

No Smart Device, separam-se as navegações de parte fixa e grid e desenha-se a tela correspondente a parte fixa, com independencia do que sucede com o grid. Isto tem consequências.

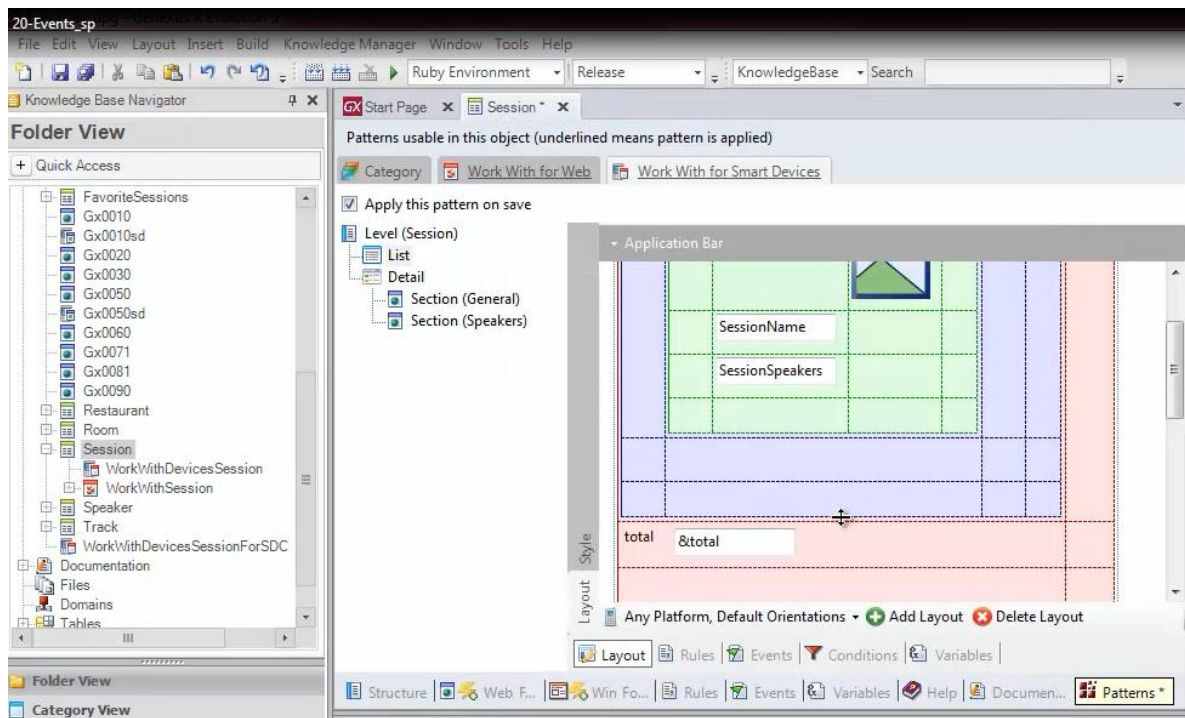


Suponhamos que desejamos mostrar no List de Sessions, a quantidade de conferências.

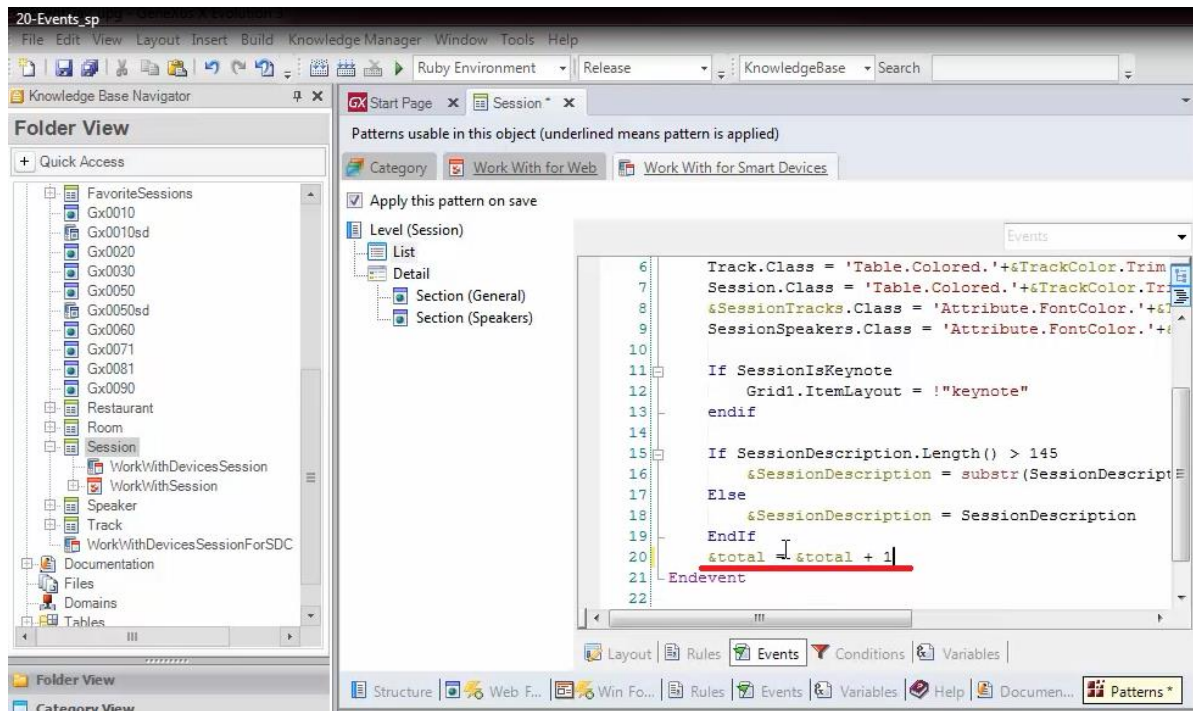
Se programarmos este panel como um web panel, inserimos uma variável &total.



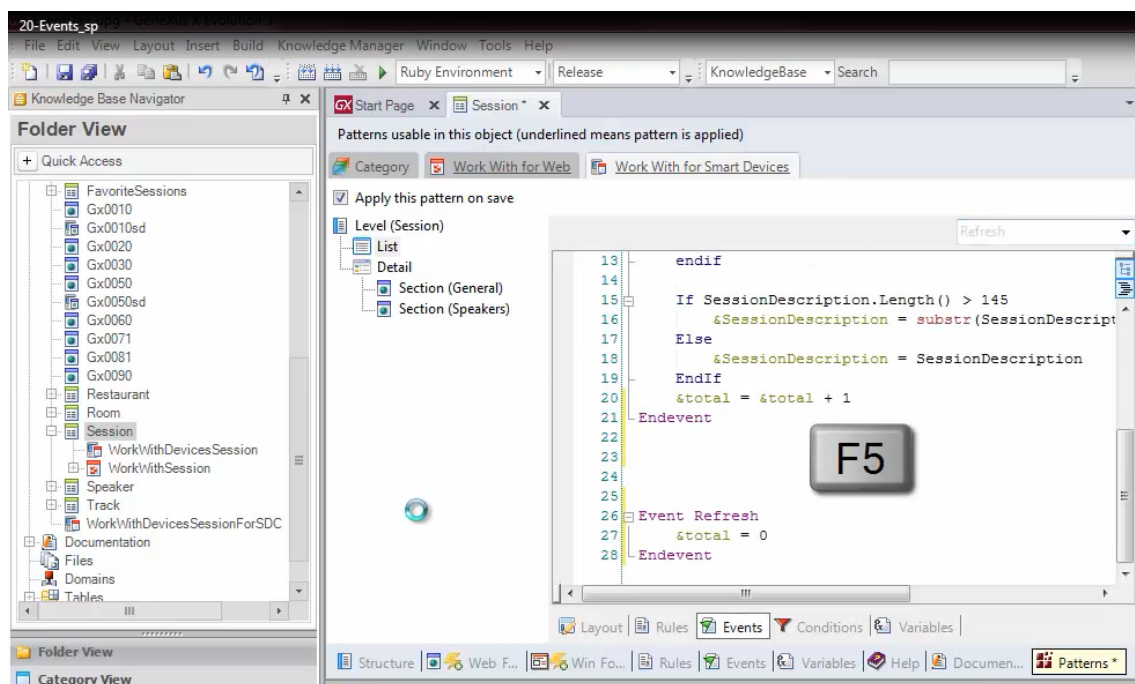
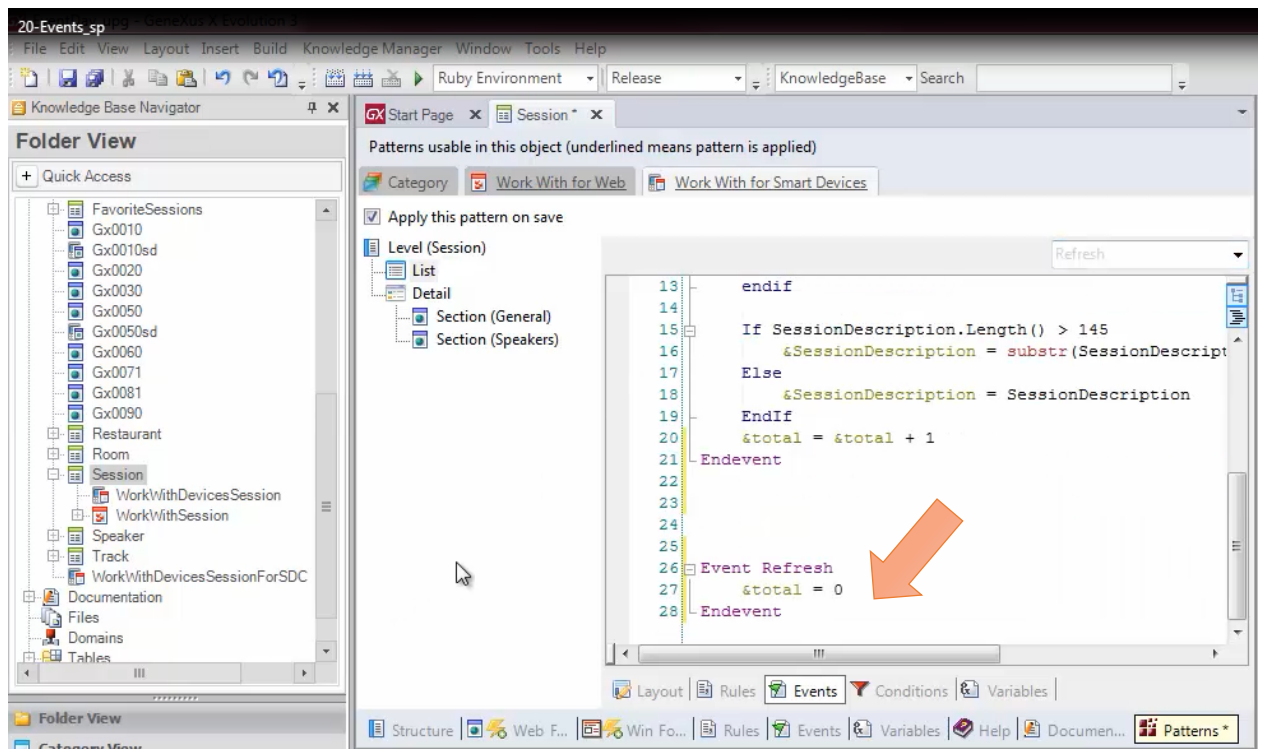
A qual seria agregada no layout fora do grid.

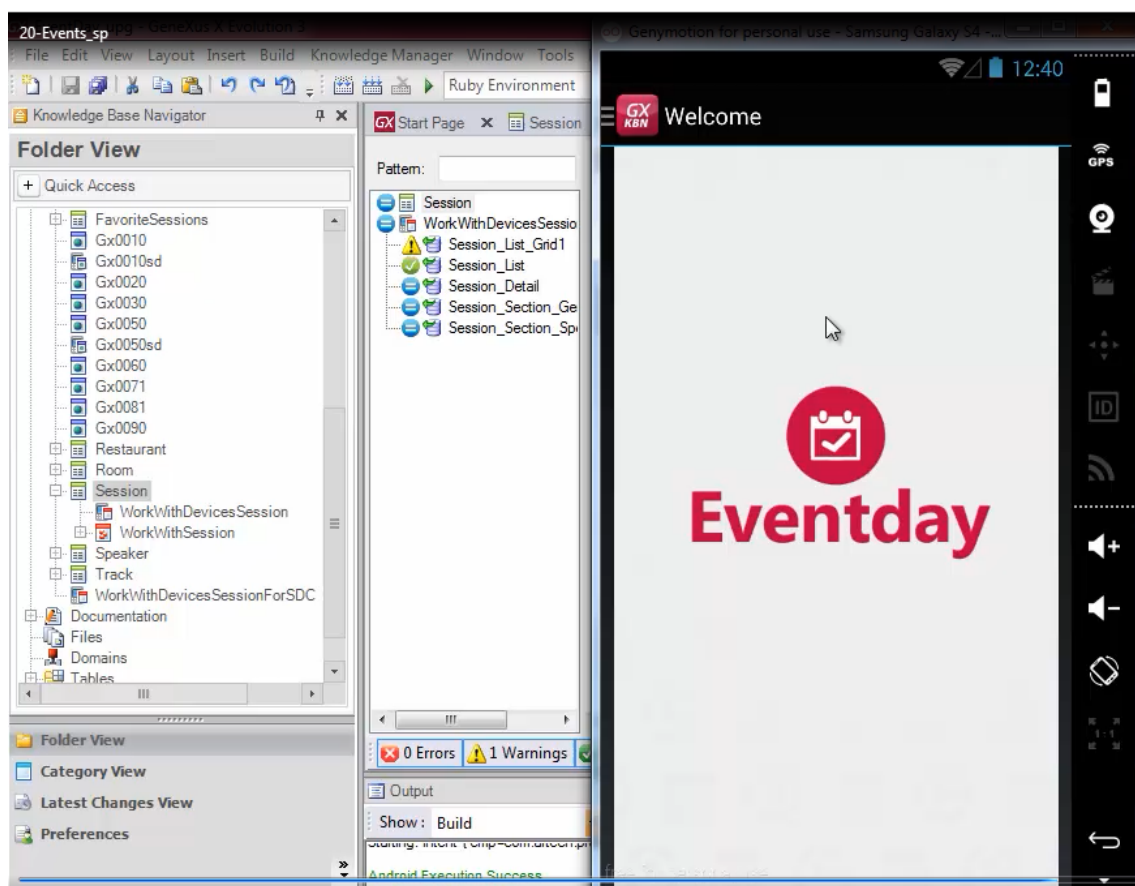
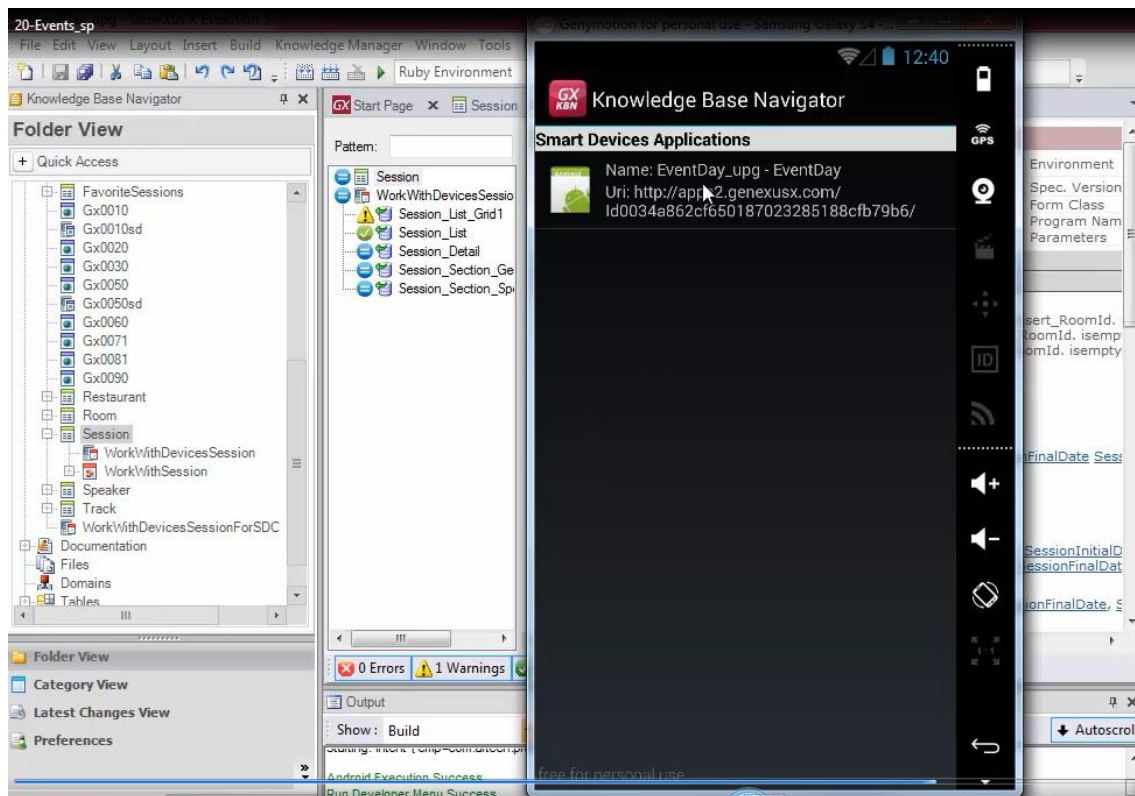


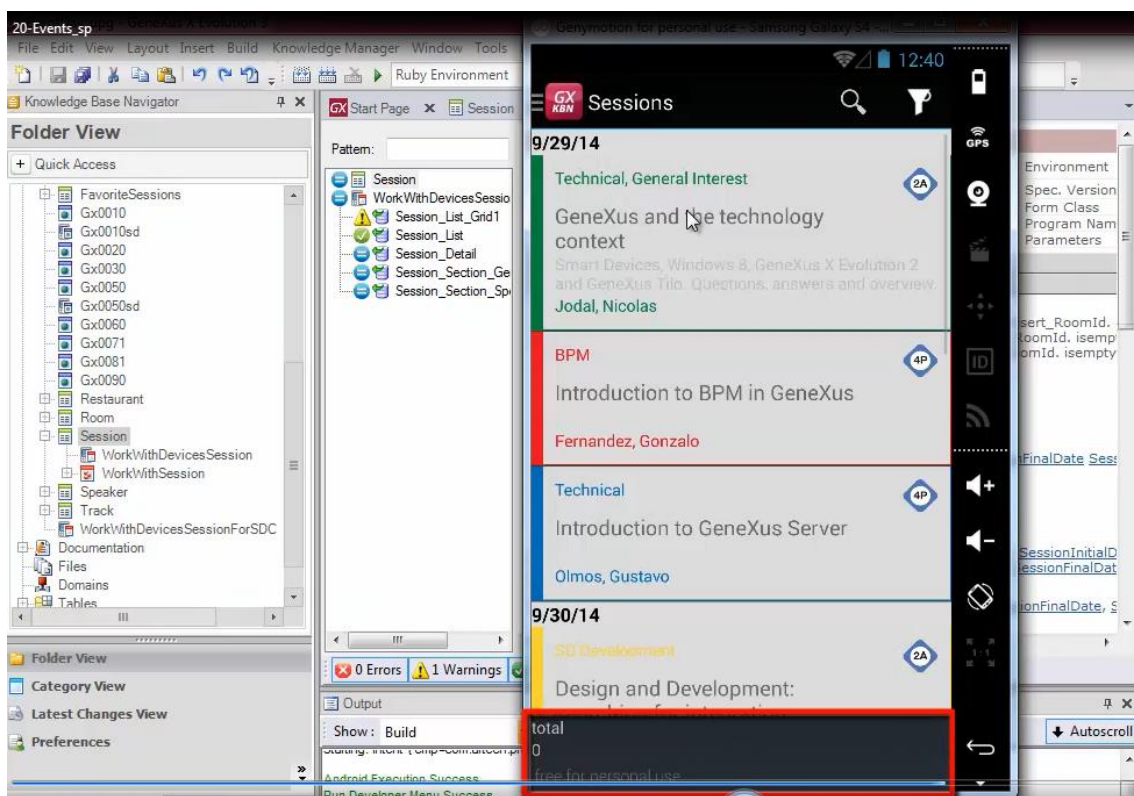
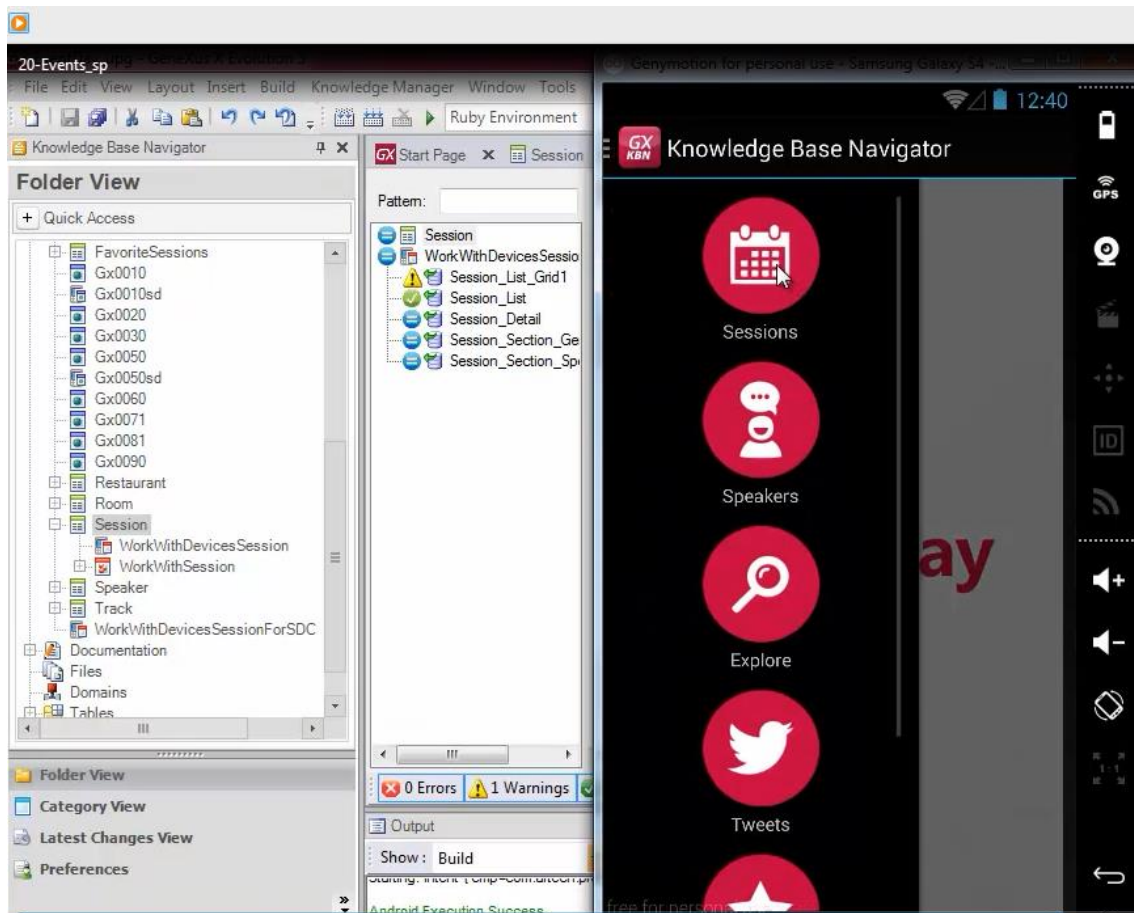
A carregáramos no evento Load, atribuindo o valor de +1.



A inicializamos no evento Refresh.

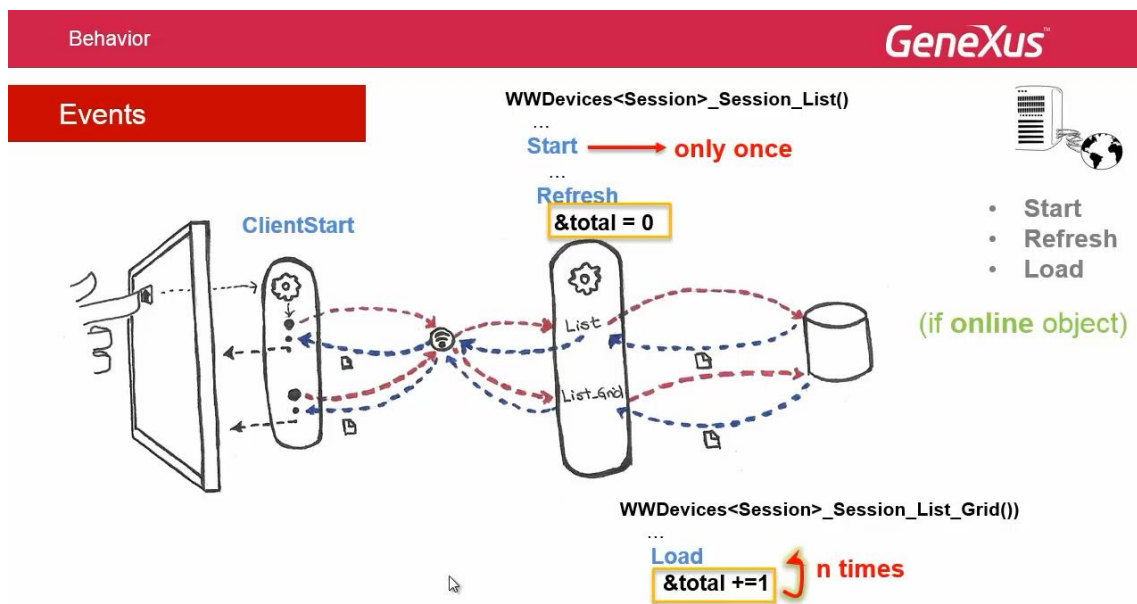




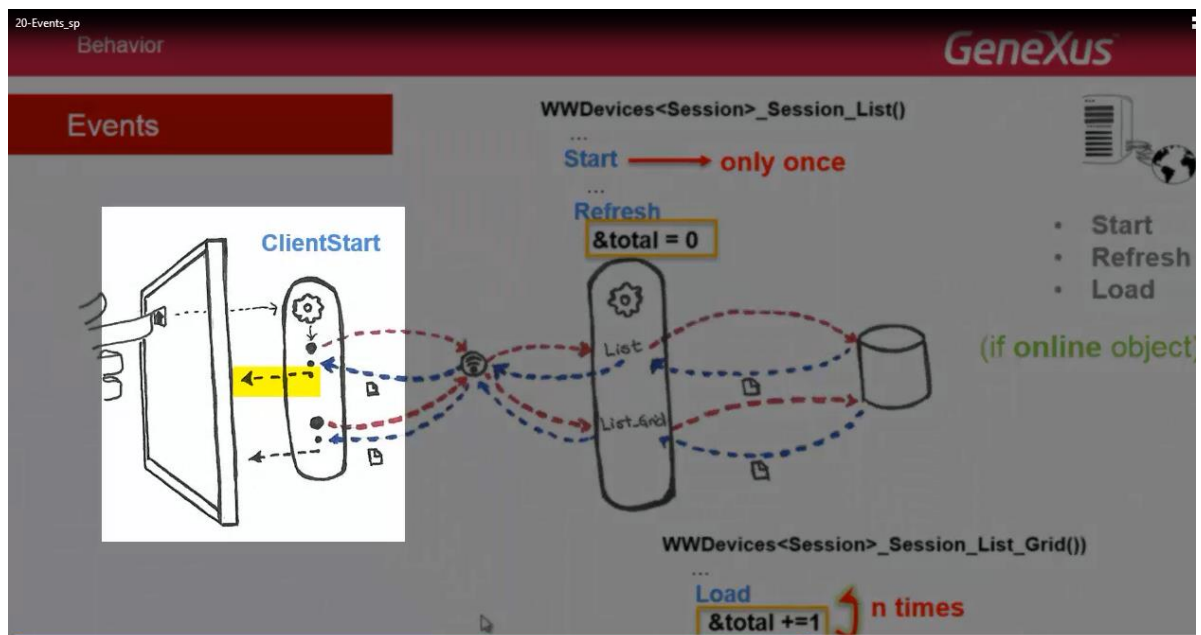


Mas... está nos mostrando zero... por quê?

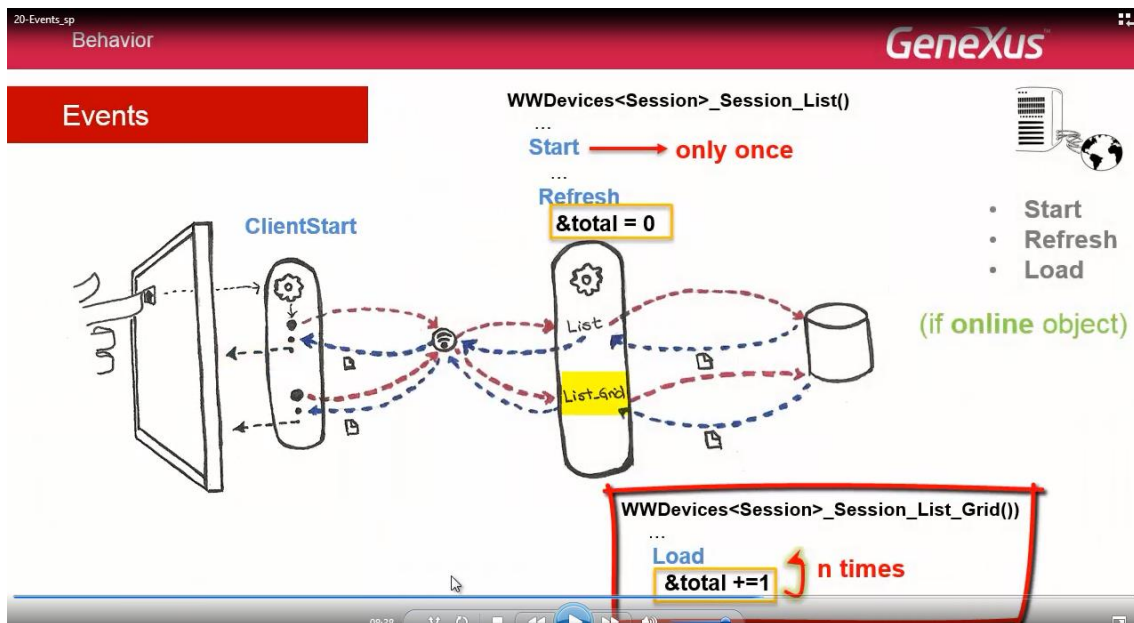
Aqui podemos ver o que fizemos.



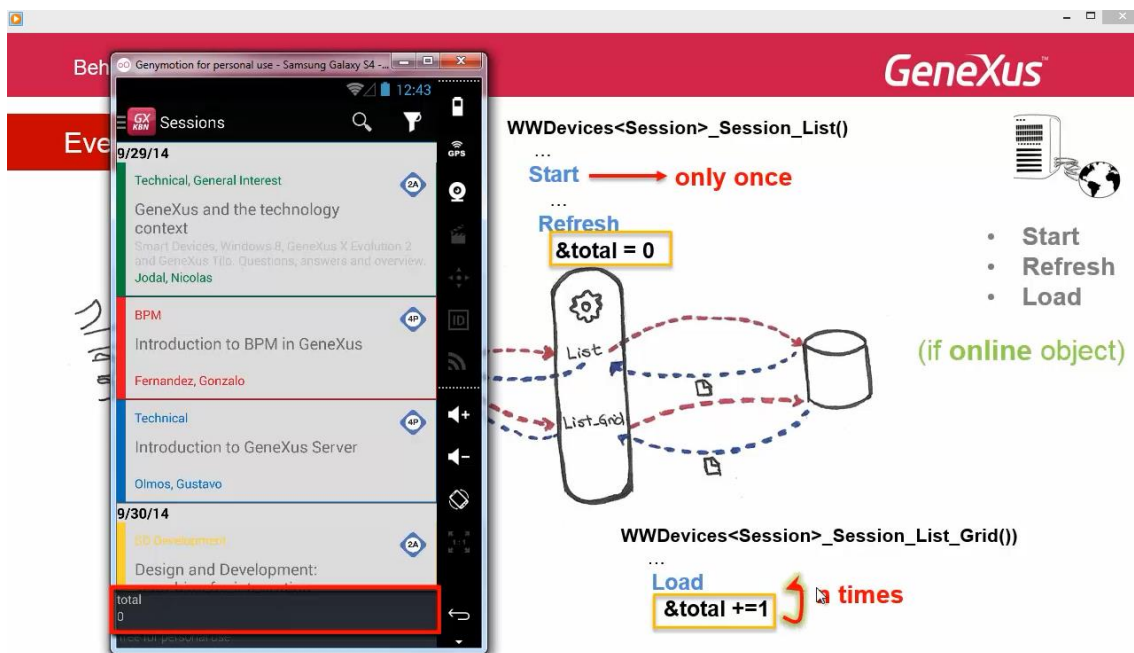
O problema é que programamos este panel como se fosse um web panel, ignorando o fato que a tela correspondente a parte fixa, será desenhada no dispositivo,



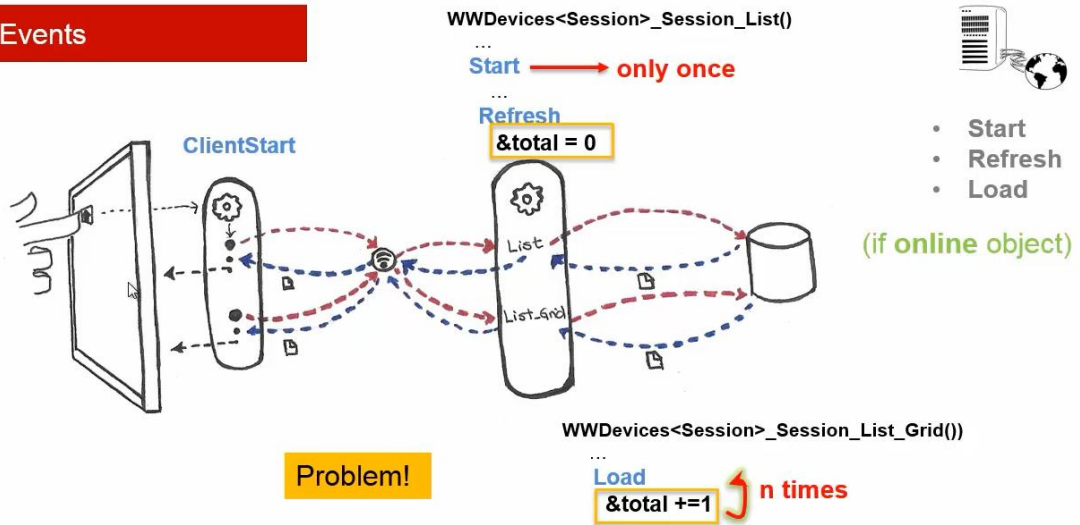
antes de chamar o data provider que devolverá as linhas do grid, para desenhá-lo na tela.



Portanto, no tempo de mostrar a variável &total, não foram chamados as N vezes do evento Load que a incrementa e muito menos será desenhado o grid...



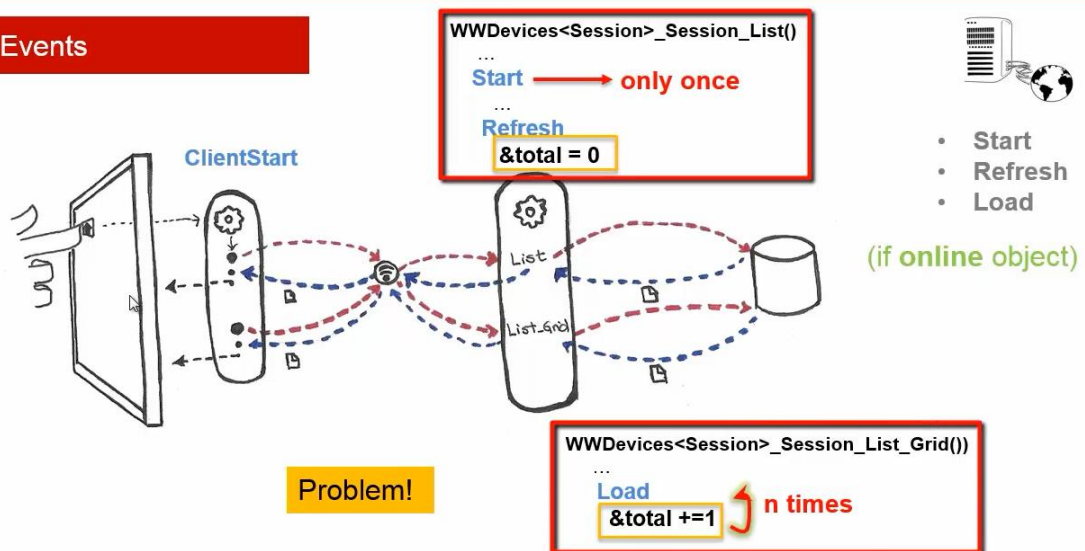
Events



Observe que esta é a razão do problema.

Os eventos Refresh e Load estão separados em 2 programas (2 data providers independentes).

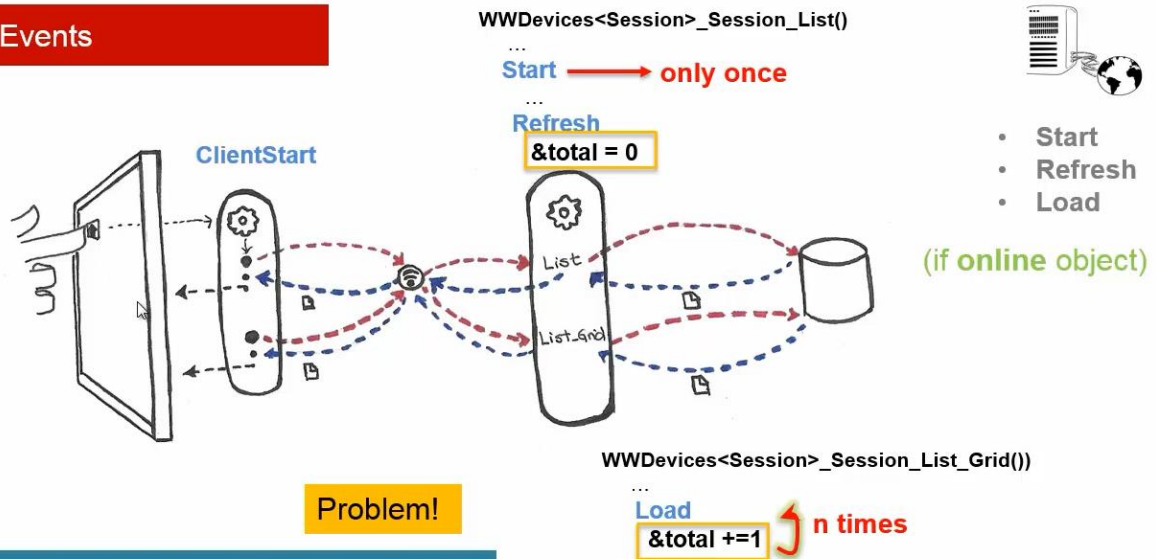
Events



Esta separação é transparente e o que é feito no Refresh será visto pelo Load.

Existe adicionalmente outro problema correspondente ao caching, podendo ser mostrado em nosso wiki.

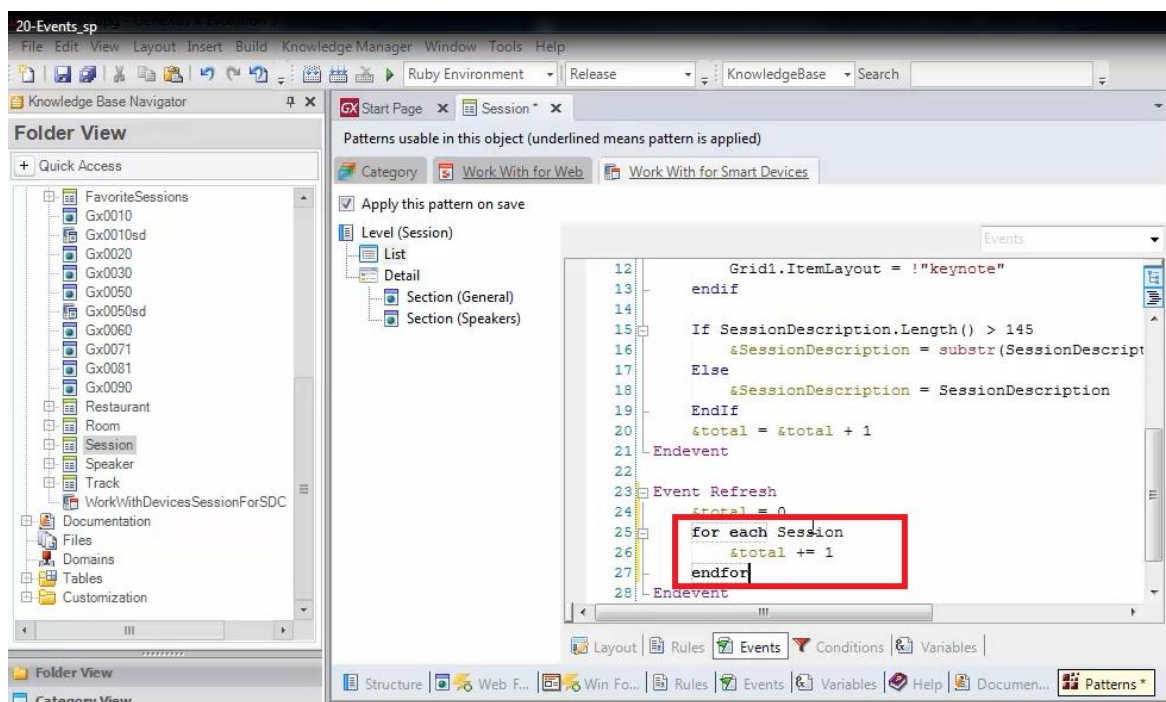
Events



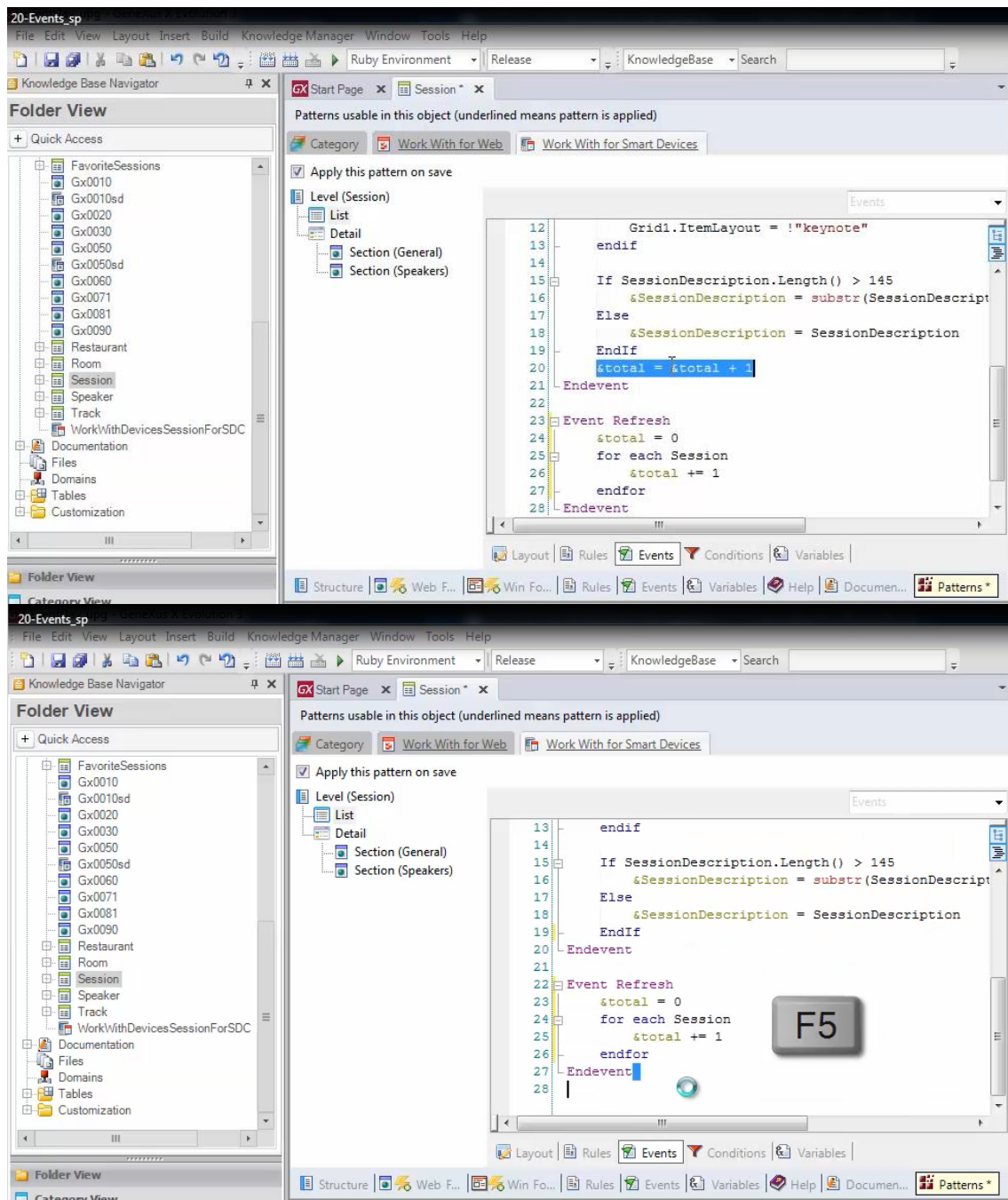
<http://wiki.gxtechnical.com/commwiki/servlet/hwikibypageid?18602>

Portanto, uma solução para conseguir mostrar o número de registro, seria dentro do evento Refresh, programando um For each que os calcule.

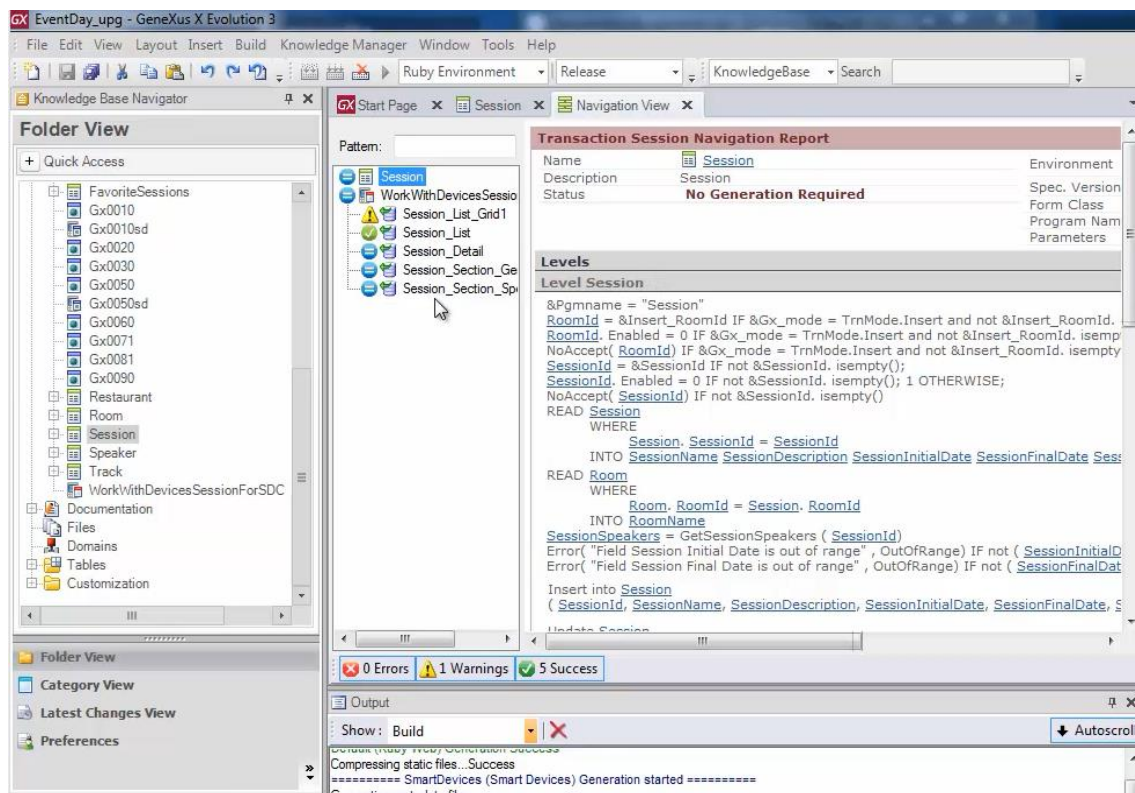
For each que terá como tabela base, a tabela associada ao primeiro nível da transação Session e para registro desta tabela, o valor da variável &total, somará 1.



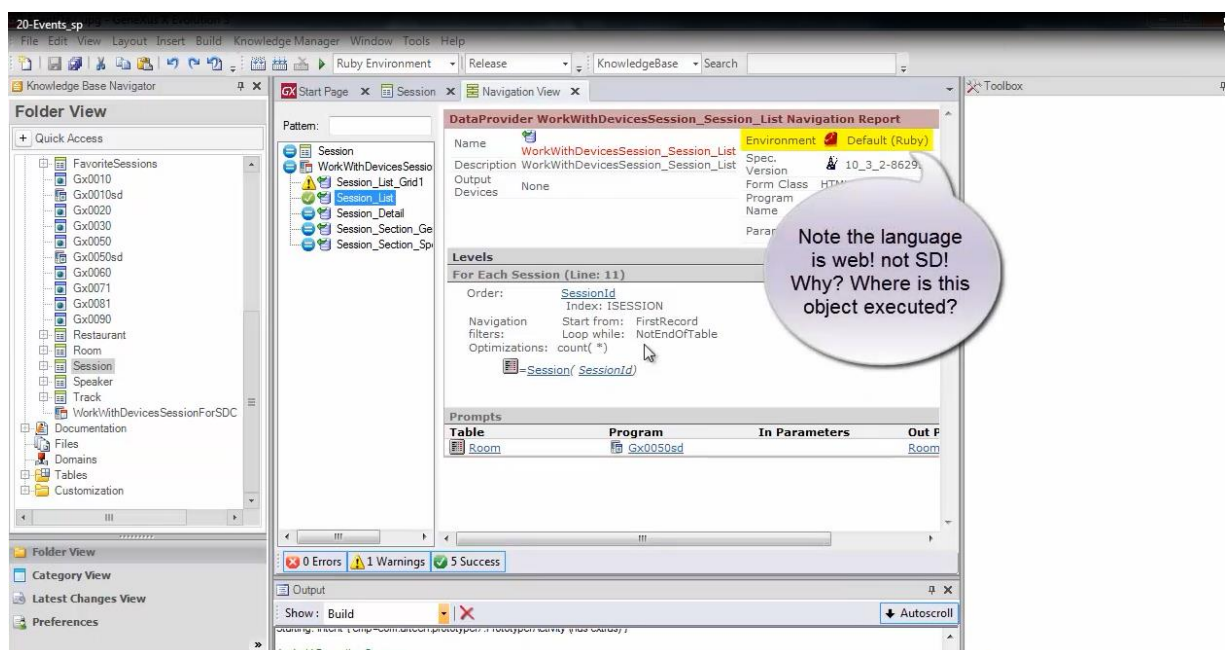
Eliminamos isto do evento Load e pressionamos F5.



Se observarmos as listagens de navegação

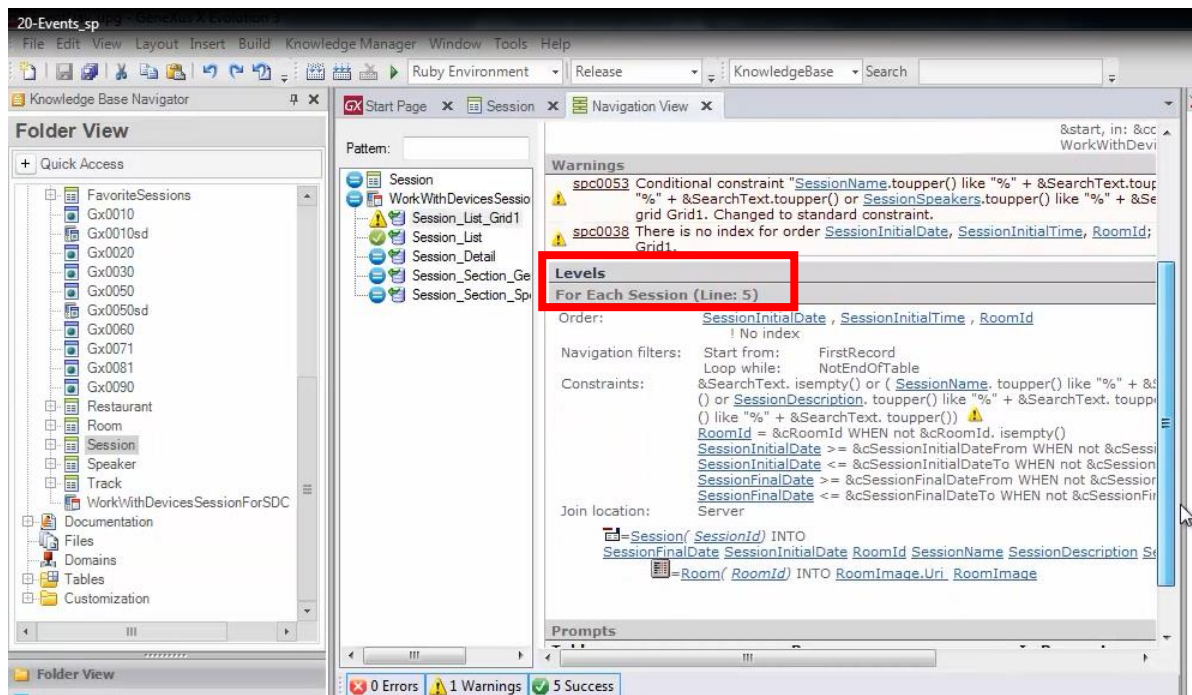


vemos que temos para o data provider Session_List correspondente, a parte fixa do List, o For each que acabamos de programar, o qual vai recorrer à tabela Session contando os registros.



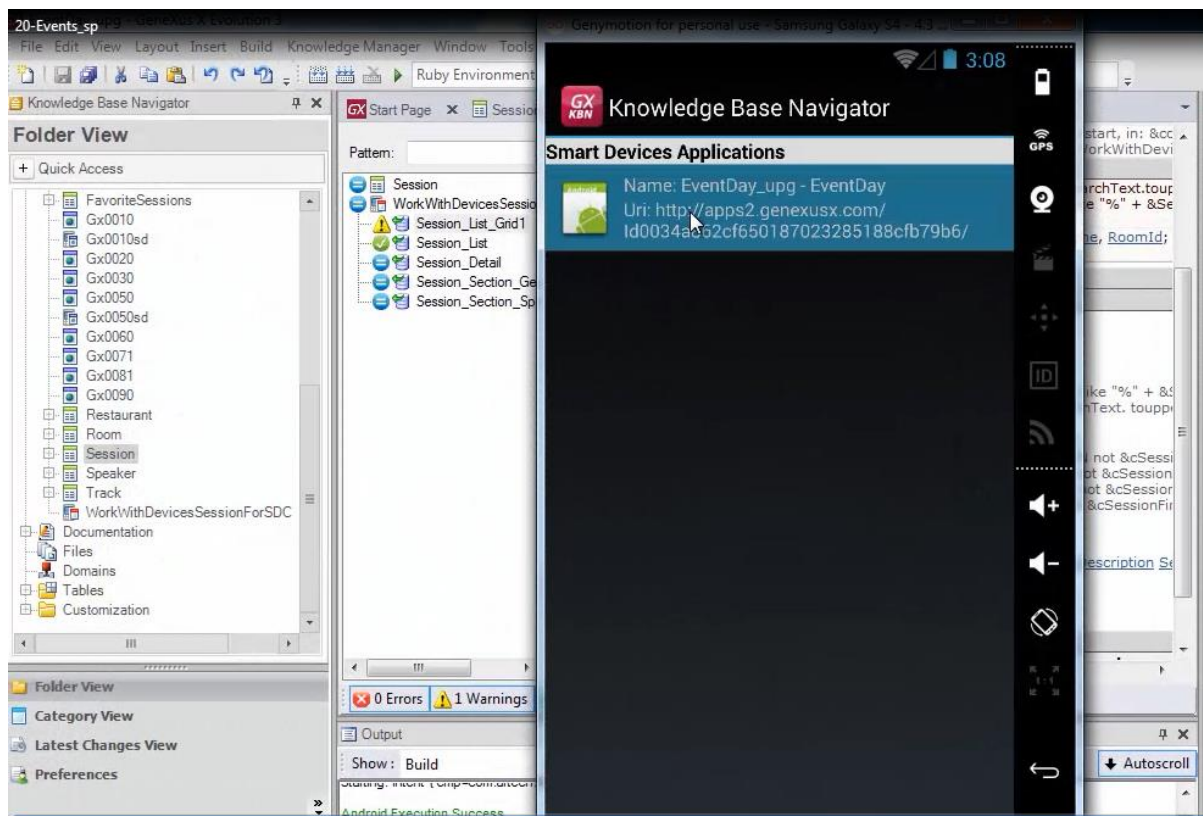
Depois de acionado este data provider, carregará a parte fixa e a variável &total terá o valor desejado.

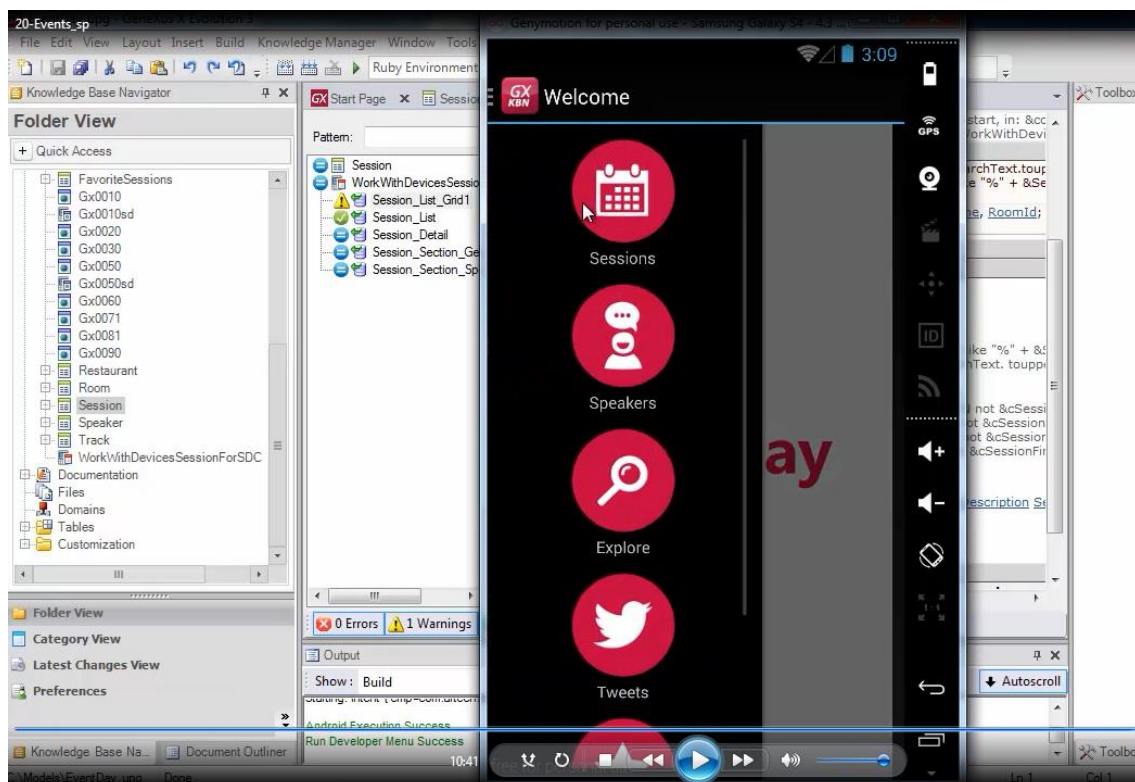
Observe o outro data provider, correspondente as linhas.



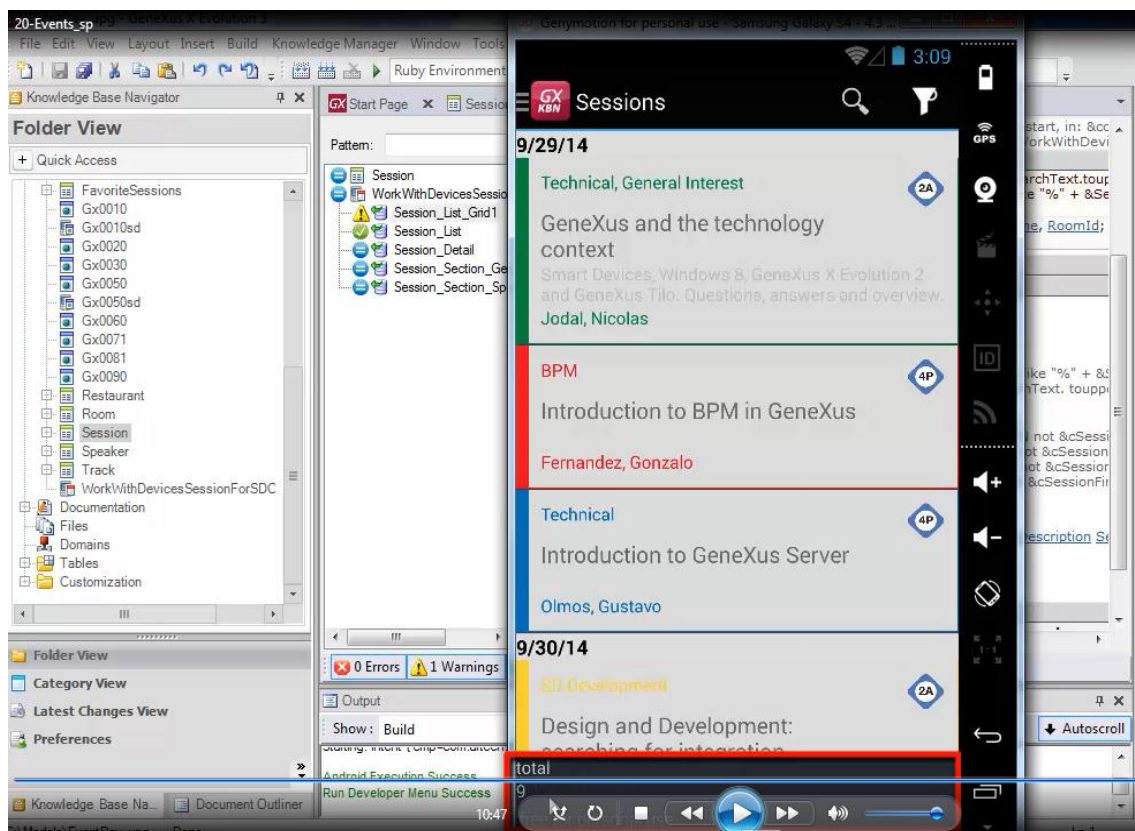
Também está sob a tabela Session.

Vamos ver em execução:

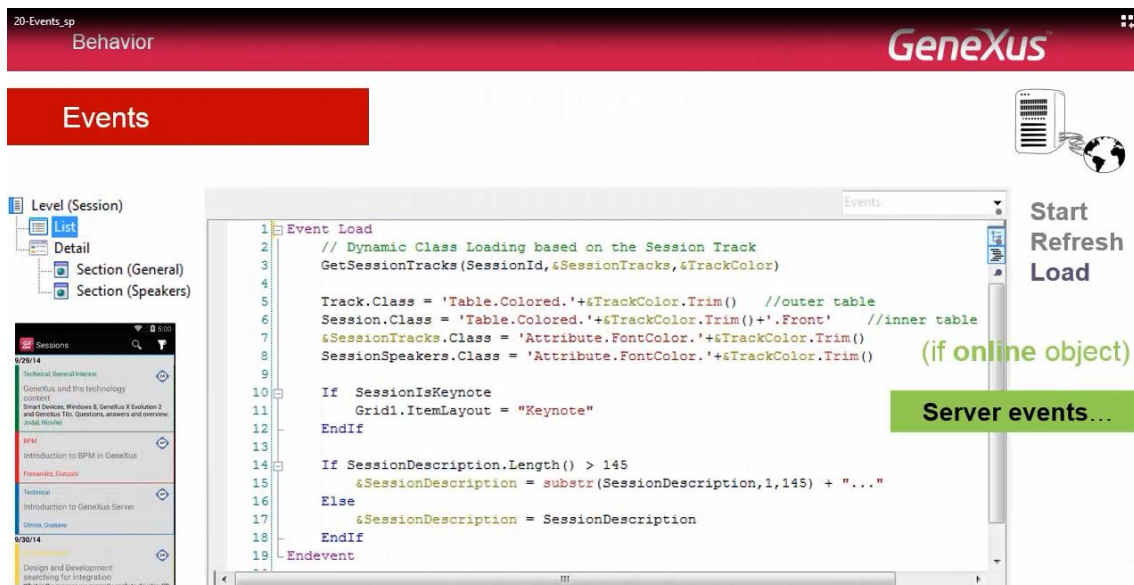




Agora a variável &total está mostrando o valor 9,

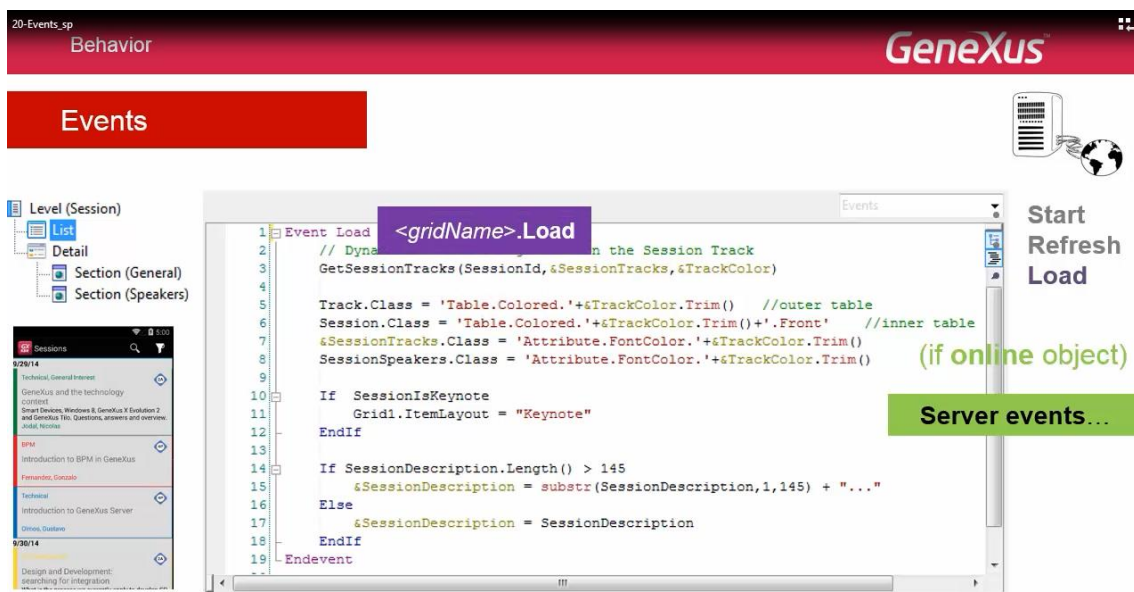


a qual coincide exatamente com a quantidade de registros contidos na tabela. O evento Load do sistema,



não requer nenhuma consideração especial, considerando que é análogo ao Load de um web panel.

Como pode-se incluir varios grids com tabela base no mesmo layout, como com web panel quando temos mais de 1 grid, será necessário especificar o Load do grid que estamos programando, escrevendo no lugar do Event Load,



Event – nome do grid – ponto – Load

Aqui mostramos o exemplo do Load para carregar os itens do grid do List

20-Events_sp Behavior
GeneXus

Events

Level (Session)

- List
- Detail
 - Section (General)
 - Section (Speakers)


```

1 Event Load
2 // Dynamic Class Loading based on the Session Track
3 GetSessionTracks(SessionId,&SessionTracks,&TrackColor)
4
5
6 Track.Class = 'Table.Colored.'+&TrackColor.Trim() //outer table
7 Session.Class = 'Table.Colored.'+&TrackColor.Trim()+'.Front' //inner table
8 &SessionTracks.Class = 'Attribute.FontColor.'+&TrackColor.Trim()
9 SessionSpeakers.Class = 'Attribute.FontColor.'+&TrackColor.Trim()
10
11 If SessionIsKeynote
12   Grid1.ItemLayout = "Keynote"
13 EndIf
14
15 If SessionDescription.Length() > 145
16   &SessionDescription = substr(SessionDescription,1,145) + "..."
17 Else
18   &SessionDescription = SessionDescription
19 EndIf
20 Endevent
          
```

Start
Refresh
Load

(if online object)

Server events...



do Work With Devices Session

Este evento executa no servidor, tendo a disposição todos os atributos da tabela estendida para serem utilizados.

Observe que o procedimento GetSessionTracks não tem que estar exposto como serviço Rest, considerando estar sendo invocado do próprio server.

20-Events_sp Behavior
GeneXus

Events

Level (Session)

- List
- Detail
 - Section (General)
 - Section (Speakers)


```

1 Event Load
2 // Dynamic Class Loading based on the Session Track
3 GetSessionTracks(SessionId,&SessionTracks,&TrackColor)
4
5
6 Track.Class = 'Table.Colored.'+&TrackColor.Trim() //outer table
7 Session.Class = 'Table.Colored.'+&TrackColor.Trim()+'.Front' //inner table
8 &SessionTracks.Class = 'Attribute.FontColor.'+&TrackColor.Trim()
9 SessionSpeakers.Class = 'Attribute.FontColor.'+&TrackColor.Trim()
10
11 If SessionIsKeynote
12   Grid1.ItemLayout = "Keynote"
13 EndIf
14
15 If SessionDescription.Length() > 145
16   &SessionDescription = substr(SessionDescription,1,145) + "..."
17 Else
18   &SessionDescription = SessionDescription
19 EndIf
20 Endevent
          
```

Start
Refresh
Load

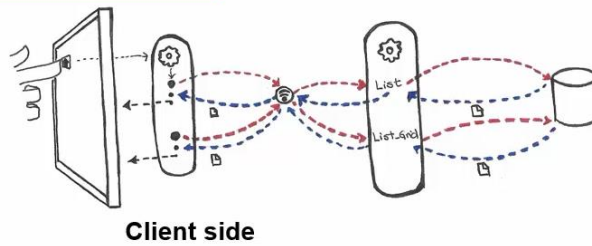
(if online object)

Server events...



Rest web service?

Events



{Flip/Split/Slide/Cascade/Tabs}.Start

Reduced
grammar

- **ClientStart**
- WW predefined events
- User Events
- **Back event**
- Control events

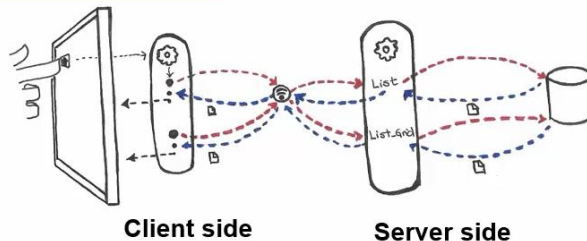
Vimos os eventos que executam no cliente e os que executam no servidor.

20-Events_sp

Behavior

GeneXus

Events



{Flip/Split/Slide/Cascade/Tabs}.Start

Reduced
grammar

- **ClientStart**
- WW predefined events
- User Events
- **Back event**
- Control events

- Start
- Refresh
- Load

If
Online

É importante diferenciá-los considerando que podemos programá-los nos eventos do cliente, seguindo uma gramática um pouco mais reduzida que podemos fazer no servidor, como veremos especificamente em um vídeo aparte.

Adiantamos que isto não afetará a implementação de uma aplicação offline. Ou seja, quanto a gramática, programará os eventos da mesma maneira se sua aplicação é online ou offline. Neste sentido é transparente.

Nos vídeos seguintes, continuaremos estudando aspectos relacionados aos eventos, como as API's para entre outras coisas, poder integrar com as funcionalidades nativas do dispositivo, a ordem de execução dos eventos, a gramática dos eventos do cliente, etc.

