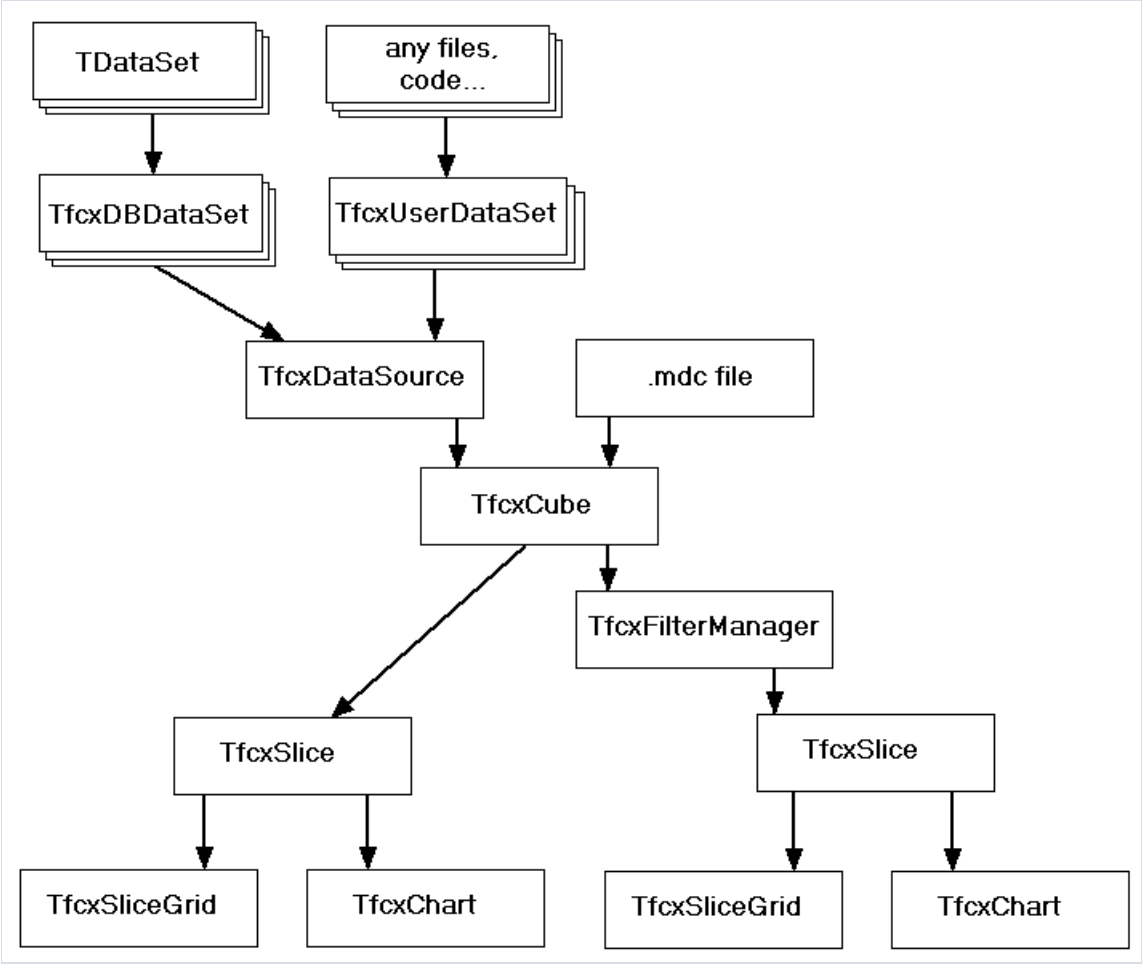
Manual de FastCube VCL Programador

Version 2021.1.4

Arquitetura do FastCube VCL

A biblioteca de componentes do FastCube VCL é um conjunto de componentes visuais e não visuais que armazenam, manipulam e visualizam dados multidimensionais. A arquitetura do FastCube 2 é representada no diagrama a seguir:



O FastCube VCL é composto dos seguintes componentes:

Componentes não visuais:

- TfcxDBDataSet - componente para conectar a uma fonte de banco de dados
- TfcxUserDataSet - componente para conectar a uma fonte do usuário (baseado em manipuladores de eventos)
- TfcxDataSource - componente que vincula todas as fontes de dados do cubo e descreve os campos e atributos
- TfcxCube - (um cubo) o armazenamento de dados principal
- TfcxSlice - (uma fatia) a estrutura responsável pela apresentação de dados e a preparação dos dados do cubo para isso
- TfcxFilterManager - gerencia a filtragem dos dados do cubo para a fatia

Componentes visuais principais:

- TfcxCubeGrid - (tabela de dados) visualiza os dados de origem do cubo
- TfcxSliceGrid - (tabela cruzada) visualiza os dados baseada em uma estrutura de fatia e permite que os usuários manipulem os dados e a estrutura
- TfcxCubeGridToolbar - (barra de ferramentas da tabela de dados) contém um conjunto de botões que

permitem executar ações na tabela de dados
- TfcxSliceGridToolbar - (barra de ferramentas da tabela cruzada) contém um conjunto de botões que permitem executar ações na tabela cruzada, na fatia e no cubo

Componentes de gráfico:

- TfcxChart - visualiza os dados de origem do cubo como um gráfico ou diagrama
- TfcxChartToolbar - (barra de ferramentas do gráfico) contém um conjunto de botões que permitem executar ações no gráfico

Salvar e carregar um cubo e uma fatia

Os dados carregados no cubo podem ser salvos para a utilização futura. Dados salvos podem ser carregados no cubo sem a necessidade de acessar o banco de dados de origem. Assim como os dados, as configurações da fatia, grupos, filtros e gráficos também podem ser salvas. Os dados e as configurações do cubo podem ser salvos em um arquivo, fluxo ou em um campo BLOB em um banco de dados, ao usar os métodos dos componentes TfcxCube, TfcxSlice, TfcxFilterManager e TfcxChart.

Dados do cubo

Dados do cubo (TfcxCube):

```
function LoadFromFile(ACubeFileName: String): Boolean;
```

Carrega os dados do cubo de um arquivo. Retorna True se o arquivo foi carregado com êxito.

O cubo é limpo antes de carregar os dados.

```
function LoadFromStream(ACubeStream: TStream): Boolean;
```

Carrega os dados do cubo de um fluxo. Retorna True se o fluxo foi carregado com êxito.

O cubo é limpo antes de carregar os dados.

```
function AppendFromFile(ACubeFileName: String): Boolean;
```

Anexa dados do cubo de um arquivo. Retorna True se o arquivo foi carregado com êxito.

O cubo mescla os dados que já contém com os dados carregados.

```
function AppendFromStream(ACubeStream: TStream): Boolean;
```

Anexa dados do cubo de um fluxo. Retorna True se o fluxo foi carregado com êxito.

O cubo mescla os dados que já contém com os dados carregados.

```
procedure SaveToFile(ACubeFileName: String; AFilter: TObject = nil);
```

Salva dados do cubo em um arquivo. Se o argumento de AFilter apontar para um objeto TfcxFilterManager, o cubo salva somente os dados que passarem pelo filtro.

```
procedure SaveToStream(ACubeStream: TStream; ACompressionLevel: TCompressionLevel = clMax; AFilter:
TObject = nil);
```

Salva dados do cubo em um fluxo. Se o argumento de AFilter apontar para um objeto TfcxFilterManager, o cubo salva somente os dados que passarem pelo filtro. Um argumento ACompressionLevel define o nível de compactação.

O cubo salva as configurações de grupos e fatias junto com os dados. O cubo não salva o estado de filtros e as configurações de gráficos vinculadas.

O arquivo do cubo tem uma extensão padrão mdc.

Exemplos de código:

```
fcxCube1.LoadFromFile('c:\cube1.mdc');
fcxCube1.AppendFromFile('c:\cube1.mdc');
fcxCube1.SaveToFile('c:\cube2.mdc');
fcxCube1.SaveToFile('c:\cube2Filter.mdc', fcxFilterManager1);
```

Configurações de fatia

Configurações de fatia (TfcxSlice):

```
function LoadFromFile(AFileName: String): Boolean;
```

Carrega as configurações de fatia de um arquivo. Retorna True se o arquivo foi carregado com êxito.

A fatia é redefinida antes do carregamento. Se as configurações carregadas possuirem informações sobre grupos, então as configurações de grupos do cubo são limpas antes do carregamento. Se as configurações carregadas possuirem informações sobre filtros, então as configurações do gerenciador de filtros são limpas antes do carregamento. Se as configurações carregadas possuirem informações sobre gráficos, então as configurações de gráfico são limpas antes do carregamento.

```
function LoadFromStream(ASliceStream: TStream): Boolean;
```

Carrega as configurações de fatia de um fluxo. Retorna True se o fluxo foi carregado com êxito.

A fatia é redefinida antes do carregamento. Se as configurações carregadas possuirem informações sobre grupos, então as configurações de grupos do cubo são limpas antes do carregamento. Se as configurações carregadas possuirem informações sobre filtros, então as configurações do gerenciador de filtros são limpas antes do carregamento. Se as configurações carregadas possuirem informações sobre gráficos, então as configurações de gráfico são limpas antes do carregamento.

```
procedure SaveToFile(AFileName: String; AStoreItems: TfcxItemsForStoreWithSlice = []);
```

Salva as configurações de fatia em um arquivo. O argumento AStoreItems define quais informações adicionais também devem ser salvas (filtros, grupos, gráficos).

```
procedure SaveToStream(ASliceStream: TStream; AStoreItems: TfcxItemsForStoreWithSlice = []);
```

Salva as configurações de fatia em um fluxo. O argumento AStoreItems define quais informações adicionais também devem ser salvas (filtros, grupos, gráficos).

Arquivos de fatias também podem conter configurações de grupos, configurações de filtro e configurações de gráfico. Arquivos de fatia são arquivos xml com uma extensão mds por padrão.

Exemplos de código:

```
fcxSlice1.LoadFromFile('c:\schema1.mds');
fcxSlice1.SaveToFile('c:\schema2.mds');
fcxSlice1.SaveToFile('c:\schema3.mds', [fcxiss_Filters, fcxiss_Groups, fcxiss_Charts]);
```

Configurações de filtro (TfcxFilterManager):

```
function LoadFromFile(AFileName: String): Boolean;
```

Carrega as configurações do filtro de um arquivo. Retorna True se o arquivo foi carregado com êxito.

As configurações do filtro são limpas antes do carregamento.

```
function LoadFromStream(AStream: TStream): Boolean;
```

Carrega as configurações do filtro de um fluxo. Retorna True se o arquivo foi carregado com êxito.

As configurações do filtro são limpas antes do carregamento.

```
procedure SaveToFile(AFileName: String);
```

Salva as configurações do filtro em um arquivo.

```
procedure SaveToStream(AStream: TStream);
```

Salva as configurações do filtro em um fluxo.

O arquivo de configuração do filtro tem uma extensão fcf por padrão.

Configurações de grupos (TfcxCube):

```
function LoadGroupsFromFile(AGroupsFileName: String): Boolean;
```

Carrega as configurações de grupos de um arquivo. Retorna True se o arquivo foi carregado com êxito.

As configurações de grupos são limpas antes do carregamento.

```
function LoadGroupsFromStream(AStream: TStream): Boolean;
```

Carrega as configurações de grupos de um fluxo. Retorna True se o arquivo foi carregado com êxito.

As configurações de grupos são limpas antes do carregamento.

```
procedure SaveGroupsToFile(AGroupsFileName: String);
```

Salva as configurações de grupos em um arquivo.

```
procedure SaveGroupsToStream(AStream: TStream);
```

Salva as configurações de grupos em um fluxo.

O arquivo de configurações de grupos tem uma extensão fcg por padrão.

Configurações de gráfico (TfcxChart):

```
function LoadFromFile(AFileName: String): Boolean;
```

Carrega as configurações de gráfico de um arquivo. Retorna True se o arquivo foi carregado com êxito.

As configurações de gráfico são limpas antes do carregamento.

```
function LoadFromStream(AStream: TStream): Boolean;
```

Carrega as configurações de gráfico de um fluxo. Retorna True se o arquivo foi carregado com êxito.

As configurações de gráfico são limpas antes do carregamento.

```
procedure SaveToFile(AFileName: String);
```

Salva as configurações de gráfico em um arquivo.

```
procedure SaveToStream(AStream: TStream);
```

Salva as configurações de gráfico em um fluxo.

O arquivo de configurações de gráfico tem uma extensão mdt por padrão.

Exemplos de código:

```
fcxFilterManager1.LoadFromFile('c:\Filter1.fcf');
fcxFilterManager1.SaveToFile('c:\Filter2.fcf');
fcxCube1.LoadGroupsFromFile('c:\Group1.fcg');
fcxCube1.SaveGroupsToFile('c:\Group2.fcg');
fcxChart1.LoadFromFile('c:\Chart1.mdt');
fcxChart1.SaveToFile('c:\Chart2.mdt');
```

Carregamento de dados

O carregamento de dados de bancos de dados e fontes do usuário.

Carregamento de um cubo a partir de uma única tabela de banco de dados

O objetivo principal da biblioteca de componentes do FastCube é criar uma tabela cruzada de resumo a partir de dados simples.

A fonte de dados do cubo mais simples é uma tabela de um banco de dados.

Para carregar dados no cubo, deve ser criada uma conexão com o banco de dados através de um descendente do componente TDataSet. A escolha exata do componente depende do componente do banco de dados usado no aplicativo.

É necessário um componente TfcxDBDataSet para vincular o descendente de TDataSet a um componente TfcxDataSource.

O componente TfcxDataSource contém a descrição completa da estrutura de dados do cubo. Ele descreve todas as fontes de dados, os arquivos das fontes, as relações entre as fontes, as regras da conversão de dados etc. Uma das fontes deve ser a fonte principal, que é atribuída à propriedade TfcxDataSource.DataSet. Somente quando todos os dados a serem carregados estiverem contidos em uma única tabela de banco de dados, a fonte principal deve ser atribuída.

Depois disso, os componentes TfcxDataSource, TfcxCube, TfcxSlice e TfcxSliceGrid devem ser vinculados juntos. TfcxCube e TfcxSlice podem ser vinculados através do gerenciador de filtros TfcxFilterManager. Se um componente TfcxFilterManager não for adicionado explicitamente ao aplicativo, TfcxSlice irá criar um TfcxFilterManager interno automaticamente. Um componente explícito de gerenciador de filtros somente é necessário quando um gerenciador de filtros é utilizado por mais de uma fatia.

TfcxDataSource pode conter uma lista dos campos de origem. Se a lista de campos não estiver presente, então todos os campos de origem serão carregados automaticamente.

TfcxDataSource.Fields contém a lista dos principais campos de origem. A lista de campos pode ser excluída ao chamar TfcxDataSource.DeleteFields. Uma lista de campos pode ser carregada da fonte ao chamar TfcxDataSource.AddFields. A lista de campos é carregada somente quando a fonte possuir objetos de campo (que foram definidos no designer de formulários ou criados automaticamente quando o DataSet foi aberto). Se a lista de campos não será alterada, então não é necessário chamar AddFields, já que ela é chamada automaticamente quando a fonte é aberta.

A fonte do cubo pode ser um componente TfcxDataSource, um arquivo de cubo ou um fluxo de cubo. O tipo de fonte a ser utilizada é especificado na propriedade TfcxCube.CubeSource : TfcxCubeSource. A enumeração TfcxCubeSource é:

```
TfcxCubeSource = (
  fccs_None,            // Nenhum
  fccs_DataSource,      // carregar de fcxDataSource
  fccs_CubeFile,        // carregar do arquivo
  fccs_CubeStream       // carregar do fluxo
  );
```

Em nosso caso precisamos usar o valor fccs_DataSource.

O cubo carrega seus dados quando o método TfcxCube.Open é chamado. O cubo abre a fonte especificada automaticamente e carrega os dados dos campos necessários.

A tabela cruzada está pronta para o uso após o carregamento dos dados.

Exemplos de código:

```
// criar componentes necessários no tempo de execução
fcxDBDataSet1 := TfcxDBDataSet.Create(Self);
fcxDataSource1 := TfcxDataSource.Create(Self);
fcxCube1 := TfcxCube.Create(Self);
fcxSlice1 := TfcxSlice.Create(Self);
fcxSliceGrid1 := TfcxSliceGrid.Create(Self);
fcxSliceGrid1.Parent := Self;
fcxSliceGrid1.Align := alClient;
// definir os vínculos entre eles
fcxDBDataSet1.DataSet := DataSet1;
fcxDataSource1.DataSet := fcxDBDataSet1;
fcxCube1.DataSource := fcxDataSource1;
fcxSlice1.Cube := fcxCube1;
fcxSliceGrid1.Slice := fcxSlice1;
// limpar a lista de campos
fcxDataSource1.DeleteFields;
// definir o tipo de fonte do cubo
fcxCube1.CubeSource := fccs_DataSource;
// carregar dados
fcxCube1.Open;
```

Configurar campos de TfcxDataSource

A lista de campos de TfcxDataSource precisa ser especificada somente nos seguintes casos:

- se somente alguns campos da fonte de dados forem necessários
- se os dados precisarem ser convertidos
- se os campos de data e hora precisarem ser divididos em partes (por exemplo dia, mês)
- se diversas fontes de dados precisarem ser vinculadas

A propriedade TfcxDataSource.Fields contém a lista de campos da fonte principal.

Os atributos do campo da fonte SourceFieldProperties depende do seu tipo - SourceFieldType : TfcxAttributeType. A enumeração de TfcxAttributeType é:

```
TfcxAttributeType = (
    fcxsft_Reference,    // um campo da fonte
    fcxsft_Custom,       // um campo de usuário
    fcxsft_Date,         // um campo de data (que pode ser separado em partes)
    fcxsft_Time          // um campo de hora (que pode ser separado em partes)
  );
```

A propriedade DataField descreve o tipo dos dados de origem, com atributos:

- nome e legenda do campo na fonte
- necessidade de converter os dados, junto ao tipo dos dados de destino
- nome e legenda do campo no cubo.

Os atributos de DataField dependem de SourceFieldType.

Exemplos de código:

```
// Carregar lista de campos
fcxDataSource1.AddFields;
// Alterar o rótulo de exibição do campo indexado em 2
fcxDataSource1.Fields[2].DataField.CubeFieldDisplayLabel := 'Customer';
// Definir uma regra para converter o campo 'Population' para uma cadeia
TfcxReferenceDataField(fcxDataSource1.Fields.FieldByName['Population'].DataField).Convert := True;
TfcxReferenceDataField(fcxDataSource1.Fields.FieldByName['Population'].DataField).CubeFieldType :=
fcdt_String;
```

Criar e ajustar atributos de TfcxDataSource

SplitProperty: TfcxSplitProperty descreve como dividir um campo baseado em seus "atributos". Atributos podem ser partes de campos de data e hora (ano, dia, hora etc.) ou os campos de uma fonte de dados que está vinculada à principal fonte de dados do cubo. Um atributo pode ter seus próprios subatributos. O nível de aninhamento de atributos não é limitado.

Um campo pode ter um atributo CaptionSourceAttribute (para substituir legendas de valores possíveis daquele atributo) e um atributo OrderSourceAttribute (para ordenar valores de acordo com a ordem dos valores dos atributos). Se esses atributos não forem definidos, então o campo irá utilizar seus próprios valores como legendas e para a ordenação.

Os atributos e o campo principal são do tipo TfcxSourceField. TfcxSplitProperty.Attributes contém a lista de atributos.

DateSplitPaths e TimeSplitPaths especificam quais partes da data e hora são necessárias para o campo (quando o campo for uma data ou hora).

Exemplos de código:

```
var
  ARefField: TfcxReferenceAttributeSFProperties;
  AAttribute: TfcxSourceField;
begin
  // carregar lista de campos
  fcxDataSource1.AddFields;
  // especificar que os atributos Day, Month e Year são necessários para o campo 'Date1'
  fcxDataSource1.Fields.FieldByName['Date1'].SplitProperty.DateSplitPaths := [odt_Day, odt_Month,
odt_Year];
  // criar um atributo para a legenda do campo 'IdClient'...
  // criar um atributo novo para o campo 'IdClient'
  AAttribute :=
TfcxSourceField(fcxDataSource1.Fields.FieldByName['IdClient'].SplitProperty.Attributes.Add);
  // definir o tipo de atributo correto
  AAttribute.SourceFieldType := fcxsft_Reference;
  ARefField := TfcxReferenceAttributeSFProperties(AAttribute.SourceFieldProperties);
  // definir a fonte do atributo como igual à fonte do campo principal
  ARefField.DataSet :=
TfcxReferenceSourceFieldProperties(fcxDataSource1.Fields.FieldByName['IdClient'].SourceFieldProperties).D
ataSet;
  // definir o nome do campo do atributo na fonte como 'FullName'
  ARefField.DataField.DataFieldName := 'FullName';
  // definir o nome do atributo criado como a fonte das legendas para o campo
  fcxDataSource1.Fields.FieldByName['IdClient'].SourceFieldProperties.CaptionSourceAttribute :=
'FullName';
  // a mesma tarefa, porém com o nome do cliente retirado de outra fonte...
  // criar um atributo novo para o campo 'IdClient'
  AAttribute :=
TfcxSourceField(fcxDataSource1.Fields.FieldByName['IdClient'].SplitProperty.Attributes.Add);
  // definir o tipo de atributo correto
  AAttribute.SourceFieldType := fcxsft_Reference;
  ARefField := TfcxReferenceAttributeSFProperties(AAttribute.SourceFieldProperties);
  // definir a fonte do atributo como fcxDataSet2 - que é uma tabela de referência que contém uma chave
'Id' e um nome 'FullName'
  ARefField.DataSet := fcxDataSet2;
  // o campo-chave na fonte do atributo é 'Id'
  ARefField.DataField.IdField.DataFieldName := 'Id';
  // O campo de nome na fonte do atributo é 'FullName'
  ARefField.DataField.DataFieldName := 'FullName';
  // definir o nome do atributo criado como a fonte das legendas do campo
  fcxDataSource1.Fields.FieldByName['IdClient'].SourceFieldProperties.CaptionSourceAttribute :=
'FullName';
end;
```

Configurar a fatia

O componente TfcxSlice é utilizado para configurar uma fatia.

Campos de fatia (TfcxSliceField) são criados automaticamente baseados nos campos do cubo.

A fatia possui contêineres que podem conter campos de área (TfcxCommonFieldOfRegion):

- XAxisContainer - eixo X, contém campos do tipo TfcxAxisField
- YAxisContainer - eixo Y, contém campos do tipo TfcxAxisField
- PageContainer - área do filtro, contém campos do tipo TfcxAxisField
- MeasuresContainer - medidas, contém campos do tipo TfcxMeasureField

Campos TfcxAxisField são criados baseados nos campos de fatia.

As medidas TfcxMeasureField podem ser criadas baseadas nos campos de fatia ou em um script FastScript.

Qualquer campo de fatia pode ser colocado em qualquer contêiner. Uma medida baseada em campos de fatia não impede que este campo seja colocado em outro contêiner ao mesmo tempo.

Criar a estrutura da fatia

Para adicionar uma dimensão (campo) a uma área de eixo ou à área de filtros, utilize os seguintes métodos de contêiner:

```
function AddDimension(ASliceField: TfcxSliceField; AName: TfcxString = ''; ACaption: TfcxString = ''):
integer;
```

Adiciona uma dimensão baseada no campo ASliceField no final da lista de campos da área especificada. Retorna a posição do campo na lista de campos da área.

Se o campo de área baseado em ASliceField já existir, então ele é movido para a posição especificada na área especificada. Se ele não existir, um novo campo de área é criado.

```
procedure InsertDimension(ASliceField: TfcxSliceField; AIndex: integer; AName: TfcxString = ''; ACaption:
TfcxString = '');
```

Insere uma dimensão baseada no campos ASliceField na posição especificada na área especificada.

Se o campo da área baseado em ASliceField já existir, então ele é movido para a posição especificada na área especificada. Se ele não existir, um novo campo de área é criado.

```
procedure DeleteDimension(AIndex: integer);
```

Exclui uma dimensão especificada pelo índice. O campo de área é destruído.

Para editar o campo Measures, utilize os seguintes métodos de contêiner:

```
function AddMeasuresField: integer;
```

Move o campo "Measures" para o final da área especificada. Retorna a posição do campo "Measures".

```
function InsertMeasuresField(AIndex: TfcxSmallCount): integer;
```

Move o campo "Measures" para a posição especificada da área especificada. Retorna a posição do campo "Measures".

```
procedure DeleteMeasuresField;
```

Exclui o campo "Measures" da área especificada. O campo "Measures" é movido automaticamente para a primeira posição da área de filtros.

IMPORTANTE!

O campo "Measures" é um campo virtual que é sempre criado mas nunca está presente na lista de campos do contêiner correspondente.

Para acessar suas propriedades, utilize a propriedade MeasuresContainer do objeto de fatia.

A posição do campo "Measures" na área é definida pela propriedade MeasuresContainer.Position. Todos os campos de área com um índice igual ou maior que MeasuresContainer.Position são exibidos após o campo "Measures".

A propriedade MeasuresContainer.Container define um contêiner que corresponde ao campo "Measures".

É melhor colocar todas as operações que alteram a estrutura da fatia entre as chamadas BeginUpdate e EndUpdate. Isso evita recálculos e recompilações desnecessárias depois de cada alteração.

Exemplo de código:

```
// iniciar a alteração da estrutura - suspender recálculos na fatia
fcxSlice1.BeginUpdate;
// adicionar um campo de fatia indexado em 0 ao eixo Y
fcxSlice1.YAxisContainer.AddDimension(fcxSlice1.SliceField[0]);
// adicionar um campo de fatia indexado em 1 à posição 0 do eixo Y
fcxSlice1.YAxisContainer.InsertDimension(fcxSlice1.SliceField[1], 0);
// adicionar um campo de fatia chamado 'FullName' ao eixo X
fcxSlice1.XAxisContainer.AddDimenssion(fcxSlice1.SliceFieldByName['FullName']);
// adicionar o campo 'Measures' ao eixo X
fcxSlice1.XAxisContainer.AddMeasuresField;
// terminar a alteração da estrutura, iniciar recálculos na fatia
fcxSlice1.EndUpdate;
```

Gerenciamento de medidas

Uma medida pode ser criada baseada em um campo de fatia ou um script FastScript.

```
function AddMeasure(ASliceField: TfcxSliceField; AName, ACaption: TfcxString; AAgrFunc: TfcxAgrFunc):
Integer;
```

Adiciona uma medida baseada em ASliceField com a função de agregação AAgrFunc. Retorna a posição da medida no contêiner.

```
function AddCalcMeasure(AName, ACaption: TfcxString; AAgrFunc: TfcxAgrFunc; AScriptFunctionName: String;
AScriptFunctionCode: TfcxString): Integer;
```

Adiciona uma medida calculada baseada na função de script AScriptFunctionName com a função de agregação AAgrFunc. AScriptFunctionCode - o código da função. Retorna a posição da medida no contêiner.

```
function AddMeasure(AField: TfcxMeasureField): Integer;
```

Adiciona a medida especificada AField. Retorna a posição da medida no contêiner.

```
procedure InsertMeasure(ASliceField: TfcxSliceField; AName, ACaption: TfcxString; AAgrFunc: TfcxAgrFunc;
AIndex: TfcxSmallCount);
```

Insere uma medida baseada em ASliceField com a função de agregação AAgrFunc na posição de contêiner especificada.

```
procedure InsertCalcMeasure(AName, ACaption: TfcxString; AAgrFunc: TfcxAgrFunc; AScriptFunctionName:
String; AScriptFunctionCode: TfcxString; AIndex: TfcxSmallCount);
```

Insere uma medida calculada baseada na função de script AScriptFunctionName com a função de agregação AAgrFunc na posição de contêiner especificada. AScriptFunctionCode - o código da função.

```
procedure InsertMeasure(AField: TfcxMeasureField; AIndex: TfcxSmallCount);
```

Insere a medida especificada AField na posição de contêiner especificada.

```
procedure DeleteMeasure(AMeasureIndex: TfcxSmallCount; ADoStopChange: Boolean = False);
```

Exclui a medida com o índice especificado.

Os métodos e propriedades MeasuresContainer permitem o acesso e a edição de medidas.

Medidas podem ser ocultadas. Medidas ocultas também são calculadas.

```
function MoveMeasure(AFromIndex, AToIndex: TfcxSmallCount): boolean;
```

Move a medida dentro do contêiner.

```
property Measures[AIndex: TfcxSmallCount]: TfcxMeasureField;
```

Propriedade de acesso à medida.

As propriedades e métodos TfcxMeasureField permitem a edição de propriedades de medidas:

```
property Visible: Boolean;
```

Medida de visibilidade.

```
property DisplayAs: TfcxDisplayAs;
```

Estilo de exibição.

É melhor colocar todas as operações que alteram a estrutura da fatia entre as chamadas BeginUpdate e EndUpdate. Isso evita recálculos e recompilações desnecessárias depois de cada alteração.

Exemplos de código:

```
// iniciar a alteração da estrutura - suspender recálculos na fatia
fcxSlice1.BeginUpdate;
// adicionar uma medida baseada no campo de fatia indexado em 3 e na função de agregação af_Sum
fcxSlice1.MeasuresContainer.AddMeasure(fcxSlice1.SliceField[3],'Sum1', 'Income', af_Sum);
// adicionar uma medida calculada que calcula a metade da receita
fcxSlice1.MeasuresContainer.AddCalcMeasure('Calc1', 'Half of Income', af_Formula, 'CalcScript1', 'Result
:= measures[''Sum1''].currentvalue / 2');
// mover a medida indexada em 1 para a posição 0
fcxSlice1.MeasuresContainer.MoveMeasure(1, 0)
// ocultar a medida indexada em 1
fcxSlice1.MeasuresContainer.Measures[1].Visible := False;
// terminar a alteração da estrutura, iniciar recálculos na fatia
fcxSlice1.EndUpdate;
```

Gerenciamento de filtros

Filtros são necessários para limitar a quantidade de dados calculados, de acordo com os critérios especificados.

É possível editar filtros usando métodos de fatia e propriedades.

Exemplos de código:

```
// limpar o filtro indexado em 3 do campo de fatia indexado em 0
fcxSlice1.SliceField[0].UVFilterOfValue[3] := False;
// iniciar a alteração do filtro
fcxSlice1.SliceFieldByName['FirstName'].BeginUpdateFieldFilter;
// desativar filtro para todos os valores do campo 'FirstName'
fcxSlice1.SliceFieldByName['FirstName'].SetNoneFilter;
// ativar o filtro com o valor 'Sergey' para o campo 'FirstName'
fcxSlice1.SliceFieldByName['FirstName'].UVFilterOfValue['Sergey'] := True;
// ativar o filtro com o valor indexado em 12 para o campo 'FirstName'
fcxSlice1.SliceFieldByName['FirstName'].UVFilterOf[12] := True;
// terminar de alterar o filtro (aplicar alterações)
fcxSlice1.SliceFieldByName['FirstName'].EndUpdateFieldFilter;
// ativar o filtro somente para o valor indexado em 4 para o campo de fatia indexado em 0
fcxSlice1.SliceField[0].UVSingleIndex := 4;
// inverter o a ativação do filtro para os valores do campo de fatia indexado em 0
fcxSlice1.SliceField[0].InverseFilter;
// configurar a ativação do filtro de acordo com o critério especificado por ARange
fcxSlice1.SliceField[0].SetRangeFilter(ARange);
// configurar o tipo do filtro como "radio"
SliceField[1].UVFilterType := uvft_Single;
```

Gerenciamento de grupos

Grupos aprimoram a representação dos dados.

Grupos podem ser editados usando os métodos e propriedades de um campo de fatia e o gerenciador de grupos (GroupManager) do campo de fatia.

Grupos podem ser editados (criados, alterados etc.) após a ativação do modo de agrupamento (CanGroup) para o campo de fatia.

Exemplos de código:

```
// ativar o modo de agrupamento
ASliceField.CanGroup := True;
// verificar se é possível usar grupos
if ASliceField.CanGroup then
begin
  // criar um grupo chamado 'Group1'
  AGroupIndex := ASliceField.GroupManager.CreateGroup('Group1').Index;
  // adicionar valores indexados em 3 ao grupo indexado em AGroupIndex
  ASliceField.GroupManager.AddUVInGroup(3, AGroupIndex);
  // adicionar os valores de 30 ao grupo indexado em AGroupIndex
  ASliceField.GroupManager.AddUVValueInGroup(30, AGroupIndex);
  // criar o grupo predefinido "Others"
  ASliceField.GroupManager.CreateOtherGroup;
end;
```