



# Desenvolvimento

© 2013 - AVMB Consultoria e Assessoria

Note:

To change the product logo for your own print manual or PDF, click "Tools > Manual Designer" and modify the print manual template.

# **Title page 1**

**Use this page to introduce the product**

---

*by -*

*This is "Title Page 1" - you may use this page to introduce your product, show title, author, copyright, company logos, etc.*

*This page intentionally starts on an odd page, so that it is on the right half of an open book from the readers point of view.  
This is the reason why the previous page was blank (the previous page is the back side of the cover)*

# Desenvolvimento

© 2013 - AVMB Consultoria e Assessoria

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: setembro 2013 in (whereever you are located)

## Publisher

...enter name...

## Managing Editor

...enter name...

## Technical Editors

...enter name...

...enter name...

## Cover Designer

...enter name...

## Team Coordinator

...enter name...

## Production

...enter name...

## Special thanks to:

*All the people who contributed to this document, to mum and dad and grandpa, to my sisters and brothers and mothers in law, to our secretary Kathrin, to the graphic artist who created this great product logo on the cover page (sorry, don't remember your name at the moment but you did a great work), to the pizza service down the street (your daily Capricciosas saved our lives), to the copy shop where this document will be duplicated, and and and...*

*Last not least, we want to thank EC Software who wrote this great help tool called HELP & MANUAL which printed this document.*

# Table of Contents

Foreword	7
<b>Part I Desenvolvimento</b>	<b>9</b>
<b>1 Arquitetura do SI*</b> .....	<b>9</b>
<b>Contexto</b> .....	<b>9</b>
Principais Métodos.....	11
<b>Aplicações Servidoras</b> .....	<b>12</b>
TBsMetaField.....	14
TBsMetaFields.....	14
TBsMetaComponent.....	14
TDmBusiness.....	17
Principais Métodos .....	17
Integração entre Objetos TDmBusiness e Objetos TBsContextoDb.....	22
Controle de Transações nos Objetos Servidores.....	23
Manipulação de Erros.....	25
Principais Métodos .....	27
<b>Catálogo</b> .....	<b>29</b>
<b>TCliBusiness</b> .....	<b>29</b>
Métodos de Atualização de Dados.....	33
Métodos de Recuperação de Dados.....	33
<b>Componentes Visuais</b> .....	<b>34</b>
TBsContainer.....	35
TBsQuantumGrid.....	36
<b>Componentes de Pesquisa (BsLocalizar)</b> .....	<b>37</b>
<b>Part II Criando uma servidora</b>	<b>40</b>
<b>1 Tabelas Físicas</b> .....	<b>40</b>
<b>2 Gerador</b> .....	<b>40</b>
<b>Predios A</b> .....	41
<b>Espacos A</b> .....	46
<b>3 Configurando o Servidor e a Type Library</b> .....	<b>47</b>
<b>4 Instalando a aplicação servidora</b> .....	<b>50</b>
<b>5 Instalando o pkn</b> .....	<b>56</b>
<b>Part III Criando uma aplicação cliente</b>	<b>60</b>
<b>1 Criando uma nova aplicação (Cadastro de Prédios)</b> .....	<b>60</b>
<b>2 Adicionando componentes ao DataModule</b> .....	<b>62</b>
<b>3 Paleta Bs Standard</b> .....	<b>63</b>
<b>4 BsLocalizar</b> .....	<b>65</b>
<b>5 Habilitando / Desabilitando controles</b> .....	<b>67</b>
<b>6 Manipulação de registros</b> .....	<b>67</b>
<b>7 Salvando e Cancelando</b> .....	<b>68</b>
<b>8 Localizar Bem Patrimonial</b> .....	<b>69</b>

<b>9 Criando métodos customizados.....</b>	<b>70</b>
Adicionado método na TLB.....	78
<b>10 Adicionado cadastro de espaços.....</b>	<b>80</b>
<b>Part IV Alterando a servidora</b>	<b>91</b>
<b>1 Redefinindo o Update.....</b>	<b>91</b>
<b>2 Exemplo de transação.....</b>	<b>98</b>
<b>Part V Criando Relatórios no SI*</b>	<b>102</b>
<b>1 QuickReport.....</b>	<b>102</b>
Leiaute .....	103
Consulta do relatório .....	110
Testando o relatório .....	115
<b>2 Relatórios RTF.....</b>	<b>117</b>
Introdução .....	117
Componentes .....	120
Consulta .....	128
Leiaute .....	129
Funções .....	132
Funções básicas.....	132
Funções avançadas.....	133
Somatórios.....	138
Outras funções.....	142
<b>Part VI Apêndice A</b>	<b>146</b>
<b>1 SGCPack.....</b>	<b>146</b>
<b>Part VII Apêndice B</b>	<b>150</b>
<b>1 TfrCPD.....</b>	<b>150</b>
<b>Métodos Virtuais .....</b>	<b>151</b>
CarregaRotulos.....	151
InitializeForm.....	152
Show AsAplicacaoChamada.....	152
Show AsEdit.....	152
Show AsNew.....	152
Show AsNone.....	152
<b>Propriedades .....</b>	<b>153</b>
RemoteSGCA.....	153
Show ingAsNew.....	153
Show ingAsEdit.....	153
NomeExecutavel.....	153
AtualizacaoEfetuada.....	153
IdAplicacao.....	153
IdUsuario.....	154
IndAuditada.....	154
<b>2 TfrCPDTool.....</b>	<b>154</b>
<b>Métodos Virtuais .....</b>	<b>155</b>
CloseDataSets.....	155
EnableDisableControls.....	155

PosicionaRegistro.....	156
SetRelParams.....	156
<b>Propriedades .....</b>	<b>157</b>
FrmState.....	157

## Part VIII Apêndice C 159

<b>1 UtilCPD.....</b>	<b>159</b>
<b>Principais Métodos .....</b>	<b>159</b>
AcrescentaZerosEsq.....	159
CheckCGC.....	159
CheckCPF.....	159
CheckPIS.....	159
FormatDateCpd.....	159
FormatDateTimeCpd.....	160
FormatTimeCpd.....	160
GetBsContextoDb.....	160
GetDescrTabCol.....	160
GetDisplayLabelPlural.....	161
GetDisplayLabelSingular.....	162
GetFormatDate.....	162
GetFormatTime.....	162
GetInClause.....	162
GetLabelCaptions.....	163
GetMensagem.....	163
GetRotulo.....	163
GetServerDate.....	164
GetServerDateStr.....	164
GetServerDateTime.....	164
GetServerDateTimeStr.....	164
GetServerTime.....	164
GetServerTimeStr.....	164
GetTempDir.....	164
IsNull .....	164
MessageDigCPD.....	165
MessageDigCPDStr.....	166
RemoteSGCA.....	166
RoundFloat.....	167
SemErro .....	167
StrToFloatDef.....	167
ValueInVarArray.....	167
VarArrayCount.....	167

## Index 0

# **Foreword**

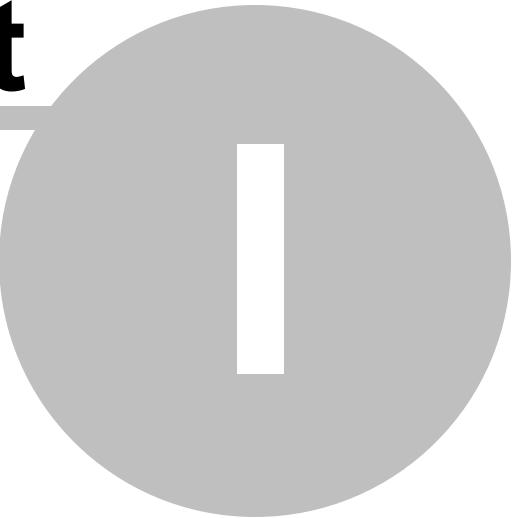
This is just another title page  
placed between table of contents  
and topics

# **Top Level Intro**

This page is printed before a new  
top-level chapter starts

# **Part**

---

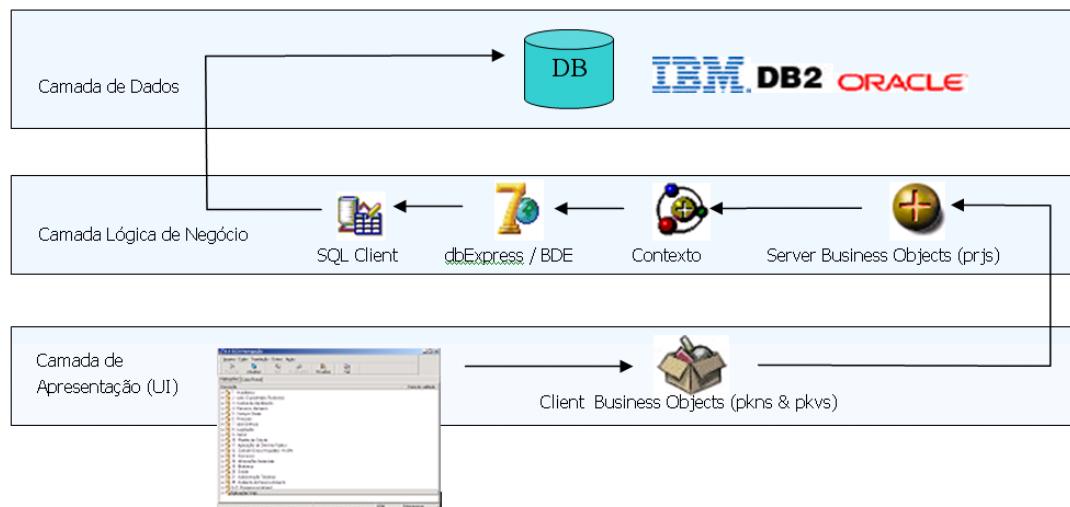


I

# 1 Desenvolvimento

## 1.1 Arquitetura do SI\*

O SI\* foi desenvolvido com base na tecnologia Multitier (multicamadas), utilizando o padrão COM. O servidor de aplicações é do tipo MTS/COM+. Os bancos de dados suportados atualmente são o Oracle e o IBM DB2. Veja a seguir um esquema da arquitetura SI\*.



**Nota:** A seguir veremos uma introdução a cada uma das partes que compõem a arquitetura SI\*.

### 1.1.1 Contexto

O contexto é um objeto COM+ que é responsável, basicamente, pela comunicação com o banco de dados. Na arquitetura de desenvolvimento do SI\* todo e qualquer acesso ao banco de dados deve ser feito através do contexto, o qual, além de disponibilizar métodos para recuperar e alterar registros do banco de dados, também possui métodos para se iniciar e finalizar transações.

O contexto contém um mecanismo interno para compartilhamento e reaproveitamento de conexões ao banco de dados, chamado *Connection Pool*, com o objetivo de otimizar o acesso ao banco de dados.

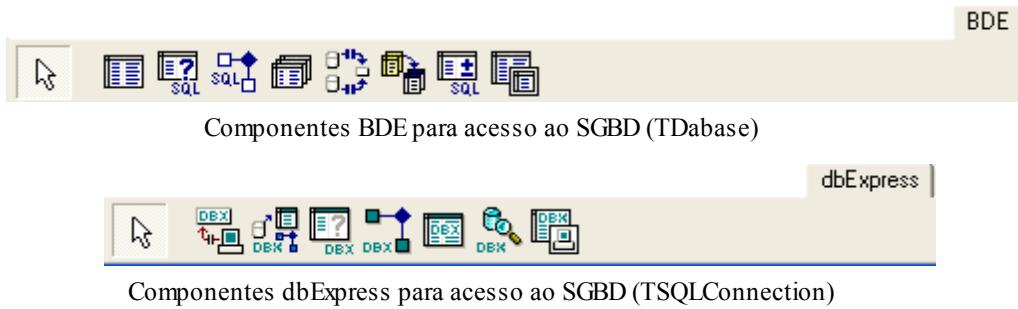
O contexto é responsável também por manter algumas caches utilizadas pelo SI\*, dentre elas a cache do catálogo (veremos o que é o catálogo do SI\* a seguir).

**Nota:** Como o contexto é o responsável pelo acesso direto ao banco de dados, é na camada onde ele estiver registrado que deve ser instalado o cliente do banco SQL e as bibliotecas do BDE ou dbExpress.

A Interface ***IBsContextoDb*** define métodos para acessar o SGBD, bem como métodos para gerenciamento de transações. Ela dá suporte também ao controle de quais usuários e aplicações estão solicitando serviços, além de suportar controle de segurança através de um sistema de chaves.

O objetivo dos objetos ***TBsContextoDb*** é fornecer acesso ao SGBD de maneira transparente, encapsulando as chamadas ao SGBD de modo a torná-las mais flexíveis. Basicamente, seu funcionamento é o seguinte: em uma área de memória compartilhada por todos os objetos ***TBsContextoDb*** são criados objetos responsáveis por efetuar a conexão com o SGBD, formando, assim, um "pool" de conexões. Quando um objeto ***TBsContextoDb*** necessita de uma conexão com o SGBD ele pega uma das conexões que estão disponíveis no "pool" e marca ela como bloqueada. Uma conexão que está bloqueada não está disponível para ser usada por outros objetos ***TBsContextoDb***. Se um objeto ***TBsContextoDb*** necessitar de uma conexão com o SGBD e não houver nenhuma conexão disponível no "pool", então ele ficará em uma fila de espera por conexões. Quando um objeto ***TBsContextoDb*** não precisa mais da conexão, ele marca a conexão que estava usando como *liberada*. Na prática, cada vez que um objeto ***TBsContextoDb*** necessita de uma conexão, ele retira uma conexão do "pool" e, quando não mais precisa dela, a devolve para o "pool" para que possa ser usada por outros objetos.

Quando o primeiro objeto ***TBsContextoDb*** é criado, são criados também objetos que serão responsáveis por efetivamente fazer a conexão com o SGBD (objetos da classe *TDatabase* para a servidora *prjContextoDB.dll* ou *TSQLConnection* para a servidora *prjContextoDBX.dll*, que fazem parte da VCL do Delphi, conforme figuras abaixo).



O número de objetos *TDatabase* / *TSQLConnection* criados é igual ao número que estiver no parâmetro *MaxConnections*. O máximo de conexões simultâneas com o SGBD é então igual a *MaxConnections*.

Se, por exemplo, *MaxConnections* for igual à vinte(20), no caso de haverem vinte e um (21) objetos ***TBsContextoDb*** ativos e a cada um for feita uma solicitação simultaneamente, então uma destas solicitações ficará em uma fila, esperando até uma conexão ser liberada.

A política de usar um limite máximo de conexões no objeto ***TBsContextoDb*** pode permitir um número grande de clientes e garantir que o limite de conexões com o SGBD não será atingido. Um cliente só está efetivamente ocupando uma conexão com SGBD quando está solicitando acesso, ou seja, durante o intervalo de tempo entre uma chamada a *InitContexto* (explicado na

próxima seção) e ou a própria liberação do objeto **TBsContextoDb**.

#### 1.1.1.1 Principais Métodos

```
InitContexto (SGCAPack: OleVariant; var Errors: ICpErrors): Integer;
```

Inicializa o contexto, associando ao objeto uma conexão com o SGBD. O tipo de dado *SGCAPack* é um *OleVariant* e é, na verdade, uma estrutura que contém informações sobre o cliente que está executando a chamada, como informações sobre o usuário e a aplicação que está sendo executada. Para maiores detalhes ver "[Apêndice A](#)"<sup>[146]</sup>.

Antes de solicitar qualquer serviço ao BsContexto, qualquer aplicação deve antes chamar *InitContexto*. Este método aloca uma das conexões do *pool*, a qual será usada durante toda a existência do objeto. Portanto, se *InitContexto* não for chamado pelo menos uma vez, todos os métodos do objeto que fazem acesso ao SGBD irão falhar.

Conforme será visto no tópico de desenvolvimento de aplicações servidoras, geralmente utilizamos o método *ConsisteBsContextoDb* declarado na *UtilCPDSrv*. Esse método instancia um objeto do tipo **TBsContextoDb** e já chama a função *InitContexto*.

```
InitDbTrans (SGCAPack: OleVariant; var Errors: ICpErrors);
```

Inicia uma transação com o SGBD. Se uma transação já estiver ativa, é incrementado um contador interno que indica que existem duas requisições de início de transação.

```
RollBackDbTrans (SGCAPack: OleVariant; var Errors: ICpErrors);
```

Finaliza com erro uma transação com o SGBD. Se o contador de transações for igual a 1 (um), ele é decrementado e a transação é finalizada, senão ele é decrementado e é setada uma variável dizendo que a transação falhou.

```
InDbTransaction (var Errors: ICpErrors; SGCAPack: OleVariant): WordBool;
```

Retorna *True* se o BsContexto está em com uma transação aberta e *False* se não estiver.

```
OpenBsQuery (MetaQueryPack, SGCAPack: OleVariant; var Errors: ICpErrors): OleVariant;
```

Executa um comando SQL de pesquisa de dados (SELECT) e retorna os dados como resultado. O comando SQL a ser executado é empacotado no parâmetro *MetaQueryPack*, que na verdade é um *OleVariant* com o seguinte formato:

Consulta SQL, Nome da Consulta, Nome da aplicação servidora, Número Parâmetros da Consulta, Nome do parâmetro n, Valor do Parâmetro n, Tipo de Dado do Parametro n.

O método executa a consulta SQL e devolve os dados empacotados em um formato proprietário do Delphi, que é interpretado pela classe *TClientDataset*, que faz parte da VCL do Delphi. O retorno da consulta deve ser atribuída a propriedade *Data* do *TClientDataset*. A classe *TClientDataset* implementa, entre outras coisas, a montagem e manipulação de um cursor.

```
OpenBsQueryRecFound(MetaQueryPack: OleVariant; out RecFound: WordBool;
SGCAPack: OleVariant; var Errors: ICpErrors): OleVariant;
```

Semelhante à *OpenBsQuery*, porém além do *OleVariant* com os dados, retorna *True* na variável de saída (**out**) *RecFound* se algum registro foi encontrado, caso contrário retorna *False*.

```
OpenBsQueryRecCount(MetaQueryPack: OleVariant; var RecordCount:
Integer; SGCAPack: OleVariant; var Errors: ICpErrors): OleVariant;
```

Semelhante à *OpenBsQuery*, porém além do *OleVariant* com os dados, retorna também o total de registros retornados pela consulta na variável *RecordCount*.

```
ExecuteBsQuery(MetaQueryPack, SGCAPack: OleVariant; var Errors:
ICpErrors);
```

Efetua um comando SQL de atualização de dados, como UPDATE, INSERT, DELETE, bem como comandos de DDL (*Data Definition Language*).

```
ExecuteBsQueryRowsAff(MetaQueryPack: OleVariant; var RowsAffected:
Integer; SGCAPack: OleVariant; var Errors: ICpErrors)
```

Semelhante a *ExecuteBsQuery*, porém além de executar comando, retorna o número de linhas afetadas pelo comando é retornado em *RowsAffected*. Se o comando a ser executado for um comando DDL, o valor de *RowsAffected* será igual a zero.

### 1.1.2 Aplicações Servidoras

Cada módulo do SI\* possui um pacote MTS/COM+ relacionado, o qual contém as aplicações servidoras que implementam as regras de negócio referentes ao módulo (existe, por exemplo, o pacote pkTributacao que contém a aplicação servidora prjTributacao, a qual implementa as regras de negócio referentes ao módulo Tributário). Essas aplicações servidoras definem e implementam uma ou, normalmente, várias Interfaces, criadas pelo editor da Type Library. A aplicação cliente (como veremos a seguir) utiliza-se dessa interface para fazer a chamada de métodos remotos.

Por padrão as servidoras do SI\* possuem uma interface para cada tabela física no banco de dados, estas interfaces devem herdar de *ICpBusiness*.

Como o Framework foi construído para atuar sobre um SGBD Relacional, é necessário

definir um esquema de mapeamento entre as tabelas físicas do SGBD e seus respectivos objetos COM. O esquema de mapeamento escolhido para ser utilizado no Framework foi o mapeamento um-para-um de tabelas para objetos. Ou seja, cada tabela do SGBD é mapeada para um objeto, que será responsável por efetuar as operações de consulta e alteração na tabela à qual ele se refere.

As operações que são comuns à maioria das tabelas, como a inserção de um registro, a busca, alteração ou exclusão de um registro por sua chave, entre outras são implementadas pelo Framework baseadas em informações advindas do Catálogo, de modo que não precisam ser programadas pelo desenvolvedor da aplicação.

Estas operações são implementadas pela classe *TDmBusiness*, que implementa a Interface *ICpBusiness*. Abaixo alguns dos métodos implementados que podem ser utilizados por todas as classes que herdam de *ICpBusiness*:

```
TDmBusiness = class(TBsServerObject, ICpBusiness, ICpBsRelationship)
protected
  function GetMetaComp(CtxSGCA: OleVariant; CtxDb: IBsContextoDb): TBsMetaComponent; overload; override;
  procedure Delete(LKValues: OleVariant; Concorrencia: Integer; CtxSGCA: OleVariant;
    const CtxDb: IBsContextoDb; var Errors: ICpErrors); virtual; safecall;
  procedure Insert(Values: OleVariant; out AutoIncGerado: integer; CtxSGCA: OleVariant;
    const CtxDb: IBsContextoDb; var Errors: ICpErrors); virtual; safecall;
  procedure Update(Values: OleVariant; out ConcorGerado: integer; CtxSGCA: OleVariant;
    const CtxDb: IBsContextoDb; var Errors: ICpErrors); virtual; safecall;
  function GetRecordPK(PkValue: Integer; NotFoundError: WordBool; CtxSGCA: OleVariant;
    const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant; virtual; safecall;
end;
```

Cabe ao programador, então, desenvolver sub-classes de *TDmBusiness* e definir e implementar as Interfaces específicas ao domínio do objeto que está desenvolvendo. Como cada objeto é, na verdade, um mapeamento de uma tabela do SGBD, o domínio do objeto deve ser, preferencialmente, referente a dados da tabela a qual ele está mapeando. Exemplo do objeto *TdmContribuinteImob* que possui as regras de negócio da tabela física CONTRIBUINTES\_IMOB:

```
type
  TdmContribuinteImob = class(TDmBusiness, IContribuinteImob)
protected
  procedure InitMetaComp(aMetaComp: TBsMetaComponent); override;
  function GetSocios(IdContribuinte: Integer; CtxSGCA: OleVariant;
    const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant; safecall;
  function GetCadastroEconomicos(IdContribuinte: Integer;
    CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant; safecall;
end;
```

Todos os métodos implementados pela classe *TDmBusiness* são escritos utilizando informações fornecidas por seu atributo *MetaComp*. Ele é um atributo de classe, ou seja, possui um valor único para todas as instâncias de objetos e é da classe *TBsMetaComponent*. É este atributo quem, de fato, faz o mapeamento das tabelas para objetos. O valor deste atributo é dado pelo método de classe *GetMetaComp*, o qual faz uma chamada ao método *InitMetaComp*, que deve ser sobreescrito pelo programador para identificar o nome da tabela que será mantida pelo objeto. Exemplo do método *InitMetaComp*:

```

procedure TdmContribuinteImob.InitMetaComp(aMetaComp: TBsMetaComponent);
begin
  with aMetaComp do
  begin
    TableName := 'CONTRIBUINTES_IMOB';
  end;
end;

```

A classe *TBsMetaComponent* é a responsável por efetuar o mapeamento de uma tabela do SGBD. Para efetuar o mapeamento, a classe *TBsMetaComponent* usa também objetos das classes *TBsMetaFields* e *TBsMetaField*. O prefixo meta à frente do nome de cada classe não é por acaso, pois pode-se dizer que estas classes contêm, na verdade, meta-dados sobre uma tabela do SGBD.

#### 1.1.2.1 TBsMetaField

A classe *TBsMetaField* é responsável por mapear uma coluna de uma tabela do SGBD e seus atributos podem ser considerados meta-dados da coluna. Alguns dos atributos da classe *TBsMetaField* são: *DisplayHint*, *DisplayLabel*, *FieldName*, *FieldType*, *LogicalKey*, *MaxValue*, *MinValue*, *MetaComponent*, *PhysicalKey*, *Required*, *Size* e *Stored*.

Além destes atributos, a classe *TBsMetaField* implementa o método *Consist*, que verifica se um determinado valor está de acordo com os atributos do *TBsMetaField*. O método *Consist* recebe como parâmetro um valor do tipo *OleVariant* e verifica se ele satisfaz todas as condições do *TBsMetaField*, como *MaxValue*, *MinValue*, *Required* e, se a coluna for do tipo *cbftCGC* ou do tipo *cbftCPF*, se o valor é um CPF ou CGC válido. Caso alguma das condições não seja satisfeita, é lançada uma exceção.

#### 1.1.2.2 TBsMetaFields

A classe *TBsMetaFields* nada mais é do que uma lista de objetos *TBsMetaField*, que implementa alguns métodos para adicionar, remover e procurar um objeto *TBsMetaField*, dentre eles os seguintes: *Add*, *Clear*, *Count*, *Delete*, *FieldByName*, *FindField* e *IndexOf*.

#### 1.1.2.3 TBsMetaComponent

A maioria dos atributos da classe *TBsMetaComponent* são *read-only* e seus valores são configurados baseados no valor da propriedade *TableName* (conforme exemplificado na seção anterior, quando foi configurado o *InitMetaComp* para a classe *TdmContribuinteImob*). Além do atributo *TableName*, podem ser informados também os valores para os atributos *SQLGetRecords*, *SQLGetRecordsLike*, *SQLGetRecordLK* e *SQLGetRecordPK*.

**TableName:** nome da tabela a qual o objeto *TdmBusiness* está mapeando.

**SQLGetRecords:** o comando SQL que será utilizado pelo método *GetRecords*, utilizado para recuperar todos os registros da tabela

**SQLGetRecordsLike:** o comando SQL que será utilizado pelo método *GetRecordsLike*, utilizado em componentes de localizar.

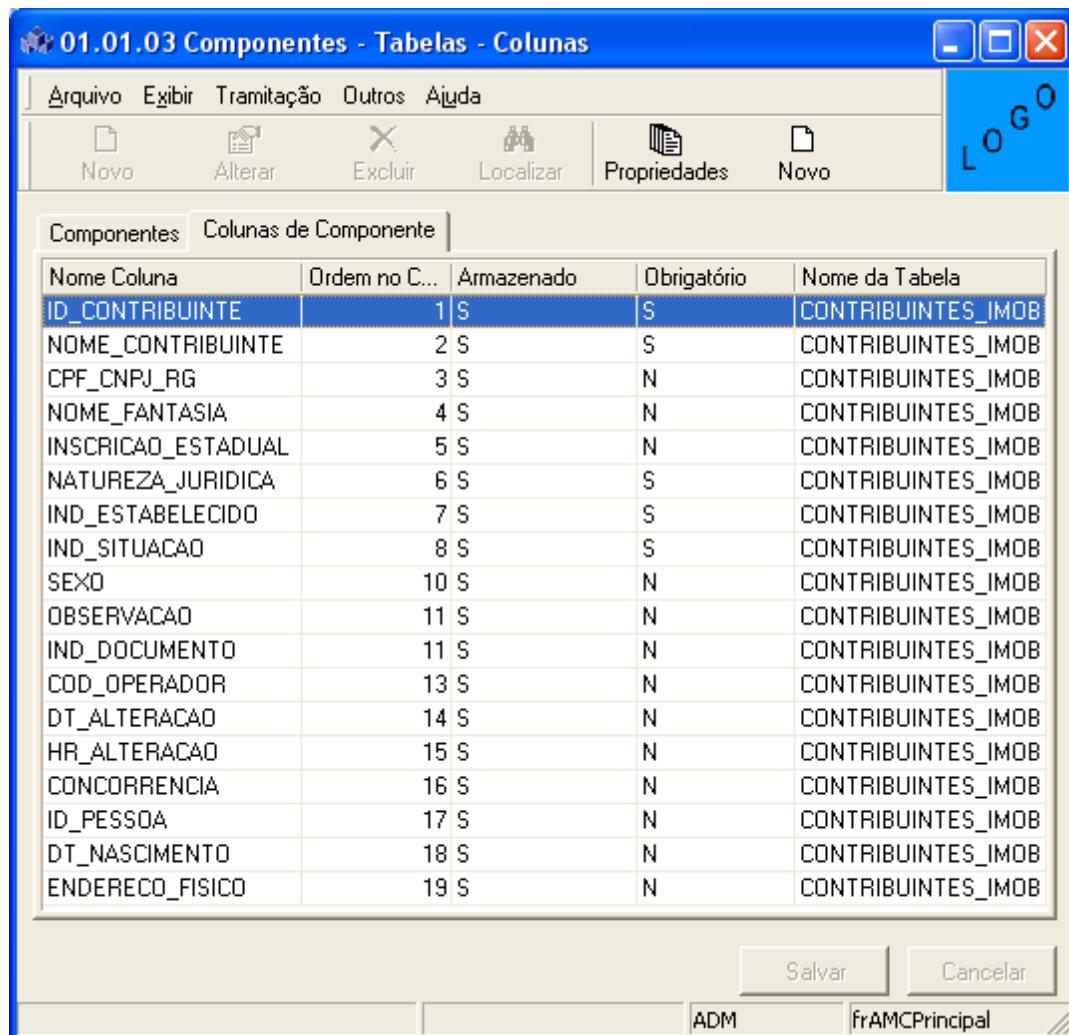
**SQLGetRecordLK:** o comando SQL que será utilizado pelo método *GetRecord*, o qual recupera um registro da tabela baseado em sua chave lógica.

**SQLGetRecordPK:** o comando SQL que será utilizado pelo método *GetRecordPK*, o qual recupera um registro da tabela baseado em sua chave física.

Exemplo de utilização:

```
procedure TdmContribuinteImob.InitMetaComp(aMetaComp: TBsMetaComponent);
begin
  with aMetaComp do
  begin
    TableName := 'CONTRIBUINTES_IMOB';
    SQLGetRecords :=
      'SELECT * FROM CONTRIBUINTES_IMOB WHERE ID_CONTRIBUINTE > 0 '+
      'ORDER BY NOME_CONTRIBUINTE';
    SQLGetRecordsLike :=
      'SELECT * '+
      'FROM CONTRIBUINTES_IMOB A '+
      'WHERE A.NOME_CONTRIBUINTE LIKE :NOME_CONTRIBUINTE '+
      'ORDER BY NOME_CONTRIBUINTE';
    SQLGetRecordLK :=
      'SELECT '+
      'A.* '+
      'FROM CONTRIBUINTES_IMOB A '+
      'WHERE '+
      'A.ID_CONTRIBUINTE = :ID_CONTRIBUINTE';
    SQLGetRecordPK := SQLGetRecordLK;
  end;
end;
```

O único atributo obrigatório é o *TableName*. Caso os outros atributos não sejam configurados, os comandos SQL serão montados dinamicamente conforme as colunas que estiverem configuradas no Catálogo de Componentes:



Os principais atributos da classe e *TBsMetaComponent* são: *AutoIncField*, *DisplayLabelPlural*, *DisplayLabelSingular*, *HasAutoIncField*, *LkFields*, *MetaQrDelete*, *MetaQrGetRecordLike*, *MetaQrGetRecordPk*, *MetaQrGetRecords*, *MetaQrInsert*, *MetaQrMaxAutoInc*, *MetaQrUpdate*, *PkField*, *SqlDelete*, *SqlGetRecordLk*, *SqlGetRecordPk*, *SqlGetRecords*, *SqlInsert*, *SqlMaxAutoInc*, *SqlGetUpdate* e *TableName*.

Além destes atributos, a classe *TBsMetaComponent* implementa o método *ConsisteValues*, cuja assinatura é:

```
ConsisteValues(Values: Variant; CtxSGCA: OleVariant; CtxDb:
BsContextoDb; var Errors: CpErrors): Boolean
```

*Values* é apenas um alias para o tipo *OleVariant*. Entretanto, ele deve ser um vetor cuja dimensão é igual ao número de campos do componente. Cada posição deste vetor deve conter um valor para um campo, e eles devem ser informados na ordem em que os campos estão na propriedade *Fields*. Este vetor é armazenado em uma *cache* no espaço de memória utilizado pelos objetos *TBsContextoDb*, onde são armazenadas informações sobre todas as tabelas do sistema. Para otimizar o acesso a esta informação, cada objeto *TdmBusiness* também mantém, em seu

espaço de memória, uma cópia das informações referentes à tabela a qual ele está mapeando.

Resumindo: Para configurar um objeto *TBsMetaComponent* para que ele mapeie uma tabela qualquer, o programador deve apenas sobreescriver o método *InitMetaComp* e informar o valor da propriedade *TableName*. Os valores para as propriedades *SQLGetRecords*, *SQLGetRecordsLike*, *SQLGetRecordLK* e *SQLGetRecordPK* podem também ser informados, entretanto, caso não o sejam, a classe *TBsMetaComponent* se encarrega de montar tais comandos utilizando-se para isso de informações do Catálogo de Componentes.

#### 1.1.2.4 TDmBusiness

Como foi descrito anteriormente, os objetos *TBsMetaComponent* são quem, de fato, efetuam o mapeamento das tabelas para objetos, guardando meta-dados a respeito das mesmas. Eles, entretanto, não são os objetos que implementam as regras do sistema. Estas são implementadas pela classe *TDmBusiness* ou por uma descendente sua.

A classe *TDmBusiness* é um objeto COM que implementa a Interface *ICpBusiness*, que define alguns métodos que são comuns à maioria dos objetos, como métodos para inserir, alterar, buscar ou excluir um registro de uma tabela do SGBD, usando para isso informações contidas em seu atributo de classe *MetaComp*.

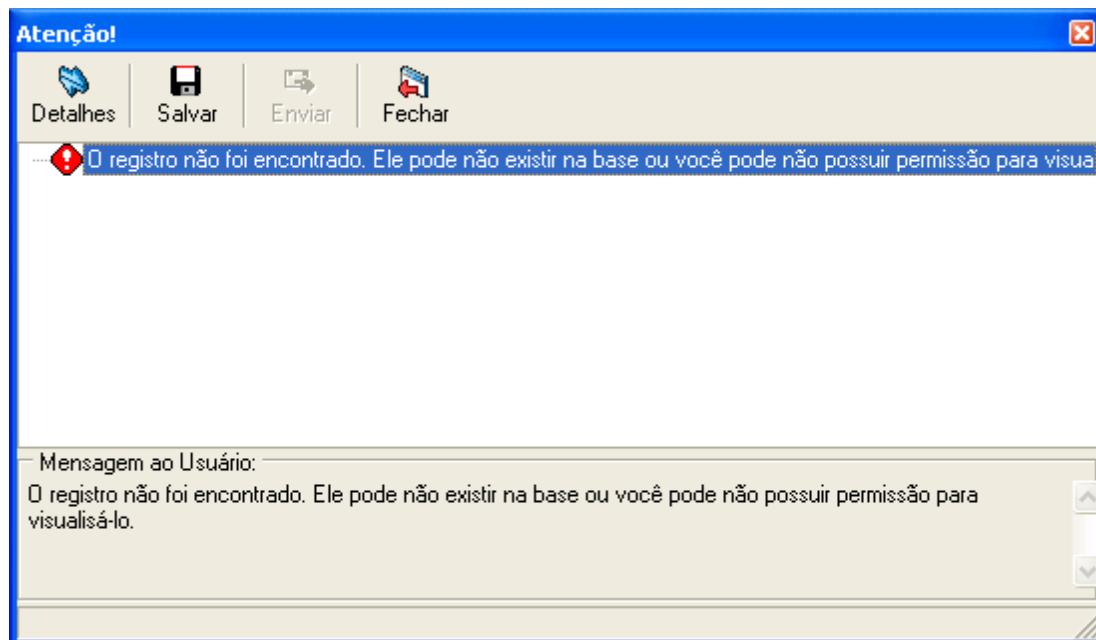
A classe *TDmBusiness* é a responsável por implementar as regras do sistema, mas ela não faz acesso direto ao SGBD. Por definição, qualquer acesso ao SGBD deve ser feito através de um objeto *TBsContextoDb*, a fim de, entre outras coisas, aumentar a portabilidade do sistema entre diferentes SGBDs.

##### 1.1.2.4.1 Principais Métodos

Todos os métodos de interface implementados pela classe *TDmBusiness* são implementados utilizando informações contidas no atributo de classe *MetaComp*. A seguir, é apresentada uma descrição de alguns destes métodos.

```
GetRecord(LKValues: OleVariant; OpenQrAux: WordBool; NotFoundError: WordBool; CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant;
```

O parâmetro *LKValues* é um *OleVariant* e é, na verdade, um vetor cuja dimensão é o número de campos que formam a chave lógica da tabela. Cada posição deste array deve conter um valor para um dos campos da chave lógica. Os valores devem ser passados na mesma ordem em que os campos estão na propriedade *LkFields* do atributo de classe *MetaComp*. Se o parâmetro *NotFoundError* for True, é gerado um erro caso nenhum registro seja encontrado, conforme tela abaixo:



A consulta executada por *GetRecord* é a que está na propriedade *SqlGetRecordLk*. Esta consulta deve trazer, obrigatoriamente, todos os campos que fazem parte do componente, incluindo os campos não-armazenados. O tipo de retorno do método, OleVariant, é um conjunto de dados que é interpretado pela classe *TClientDataset*. Os parâmetros *CtxSGCA*, *CtxDb* e *Errors* são comuns em quase todos os métodos da classe *TDmBusiness*. *CtxSGCA* é explicado no [Apêndice A](#)<sup>146</sup>. *CtxDb* é explicado na seção sobre [controle de transação](#)<sup>23</sup>. *Errors* é explicado na seção sobre [manipulação de erros](#)<sup>25</sup>.

```
GetRecords(NotFoundError: WordBool; CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant
```

Executa a consulta contida *SqlGetRecords* do atributo *MetaComp*. Esta consulta deve trazer todos os registros da tabela. Se o parâmetro *NotFoundError* for True, gera um erro caso nenhum registro seja encontrado.

```
GetRecordPK(PkValue: Integer; NotFoundError: WordBool; CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant
```

Semelhante ao método *GetRecord*. Executa porém a consulta contida no atributo *SqlGetRecordPk* do atributo *MetaComp*. Esta consulta deve trazer, obrigatoriamente, todos os campos que fazem parte do componente, incluindo os campos não-armazenados. Só tem sentido se a chave física da tabela não for composta, ou seja, possuir somente um campo. Caso contrário, este método irá falhar. Porém, a grande maioria das tabelas do SI\*, possui algum campo definido como chave física.

```
Insert(Values: OleVariant; out AutoIncGerado: integer; CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors)
```

O parâmetro *Values* é um OleVariant e é um vetor com valores para todos os campos do

componente. Os valores que são passados em *Values* serão inseridos na tabela. Antes disto eles são consistidos no método *ConsisteInsert* usando o método *ConsisteValues* de *MetaComp*. O registro só é inserido se o método *ConsisteInsert* retornar True. O método *ConsisteInsert* pode ser sobreescrito pelo programador quando for necessária alguma consistência extra antes da inserção de um registro. Por exemplo, na servidora *TdmContribuinteImob*, o método *ConsisteInsert* pode ser sobreescrito para adicionar uma verificação adicional, se foi informado CPF ou CGC para o contribuinte, caso contrário, retornar False para impedir a inclusão do registro. Caso o componente possua um campo do tipo *cbftAutoInc*, o novo valor gerado para este campo é retornado em *AutoIncGerado*. O comando SQL usado para fazer o *Insert* no SGBD é o que está na propriedade *SqlInsert* do atributo *MetaComp*, e é montado baseado em informações advindas do Catálogo.

```
Update(Values: OleVariant; out ConcorGerado: integer; CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors)
```

Executa o comando SQL da propriedade *SqlUpdate* do atributo *MetaComp* para alterar um registro do SGBD. Além disso, é executado um código para controlar a concorrência de duas alterações simultâneas ao mesmo registro, através da coluna "Concorrencia", campo que deve existir em todas as tabelas do SI\*. Este controle é efetuado através do valor deste campo. Para alterar um registro, o valor informado para o campo "Concorrencia" deve ser igual ao que está no SGBD. Se o valor for menor que o que está no SGBD, isto significa que este registro já foi alterado por outro cliente e um erro é gerado. Se o registro não for encontrado no SGBD, então é porque ele já foi excluído, e isto também gera um erro. O parâmetro *ConcorGerado* é um parâmetro de saída que recebe o valor que está no SGBD para o campo "Concorrencia". Antes de efetuar a alteração do registro, os valores passados em *Values* são consistidos no método *ConsisteUpdate* usando o método *ConsisteValues* de *MetaComp*. O registro só é alterado se este método retornar True. Para consistências mais específicas, o programador deve sobrecrever o método *ConsisteUpdate*.

```
Delete(LKValues: OleVariant; Concorrencia: Integer; CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors)
```

Exclui um registro do SGBD baseado em sua chave lógica. O comando SQL executado é o que está na propriedade *SqlDelete* de *MetaComp*. O registro só será excluído se o método *ConsisteDelete* retornar o valor True. O mesmo controle de concorrência efetuado no método *Update* é efetuado também no método *Delete*.

```
MultiInsert(TableName, KeyFieldName: string; FieldNames, FieldTypes, MultiValues: OleVariant; GenKeys: Boolean; var Keys: OleVariant; CtxSGCA: OleVariant; CtxDB: IBsContextoDb; var Errors: ICpErrors)
```

Em algumas situações é necessário inserir vários registros no SGBD de uma única vez. Um exemplo seria inserir as parcelas de um parcelamento. Se o parcelamento foi feito em 120 parcelas, serão inseridas 120 linhas na tabela *Parcelas\_Tributos*. Chamar 120 vezes o método *Insert* funciona, porém não é eficiente. A solução para essa situação é utilizar o método *MultiInsert*, no qual os parâmetros são explicados a seguir:

**TableName**: nome da tabela no qual serão inseridos os dados.

**KeyFieldName:** nome da coluna que é chave física da tabela. Se a tabela somente possuir chave lógica, esse parâmetro deve ser informado como uma string vazia.

**FieldNames:** OleVariant, é um vetor onde cada posição corresponde ao nome das colunas da tabela. Os campos obrigatórios nas tabelas do SIM (Cod\_operador, Dt\_Alteracao, Hr\_alteracao, Concorrencia e Endereco\_Fisico) não devem constar nesse vetor, pois serão incluídos posteriormente pelo método. Se o parâmetro *KeyFieldName* foi informado, ele não deve constar nesse vetor.

**FieldTypes:** OleVariant, é um vetor onde cada posição corresponde ao tipo de dado da coluna. A ordem dos elementos no vetor deve corresponder a ordem das colunas contida no parâmetro *FieldNames*.

**MultiValues:** OleVariant, é um vetor com várias linhas, onde cada linha conterá os valores das colunas informadas no parâmetro *FieldNames*. No exemplo das parcelas, *MultiValues* terá 120 linhas.

**GenKeys:** Deve ser informado True para o *KeyGenerator* gerar os valores da chave física das linhas a serem incluídas. Nesse caso, obrigatoriamente a tabela a ser inserida deve estar no Catálogo. Para o caso de tabelas que só possuem chave lógica, o parâmetro deve ser False, pois a tabela não possui chaves a serem gerados.

**Keys:** Variável de saída. Caso *GenKeys* for True, nessa variável estarão todos os valores de chaves gerados pelo *KeyGenerator*.

Exemplo de utilização:

```

procedure TdmCCorrenteFiscal.ExemploMultiInsert(IdCcorrenteFisc: Integer;
                                                CtxSGCA: OleVariant; const CtxDb: IBsContextoDb;
                                                var Errors: ICpErrors): OleVariant;
var FieldNamesCC, FieldTypesCC, ValuesPT, MultiValuesPT, KeysPT: OleVariant;
    i: integer;
begin
try try
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    //Preenche os dados da variável FieldNames
    FieldNamesPT := VarArrayOf(['ID_CCORRENTE_FISC', 'NUM_PARCELA', 'DT_VENCIMENTO',
                                'VL_PARCELA', 'MOEDA_TAB', 'MOEDA_ITEM', 'SIT_PGTO_TAB', 'SIT_PGTO_ITEM']);
    //Preenche os dados da variável FieldTypes
    FieldTypesPT := VarArrayOf([ftInteger, ftInteger, ftDate,
                                ftFloat, ftInteger, ftInteger, ftInteger, ftInteger]);
    //Cria o laço de repetição para preencher os valores a serem inseridos na
    //tabela PARCELAS_TRIBUTOS
    for i:=1 to 120 do
begin
    begin
        ValuesPT := VarArrayOf([IdCcorrenteFisc, NumParcela, DtVencimento,
                               VlParcela, MOEDA_TAB, MoedaItem, SIT_PGTO_TAB, SitPgtoItem]);
        VarArrayRedim(MultiValuesPT, VarArrayHighBound(MultiValuesPT, 1)+1);
        //Adiciona valores para a variável MultiValues
        MultiValuesPT[VarArrayHighBound(MultiValuesPT, 1)] := ValuesPT;
    end;
    //Envia os dados para serem inseridos no SGBD
    MultiInsert('PARCELAS_TRIBUTOS', 'ID_PARCELA', FieldNamesPT, FieldTypesPT,
                MultiValuesPT, True, KeysPT, CtxSGCA, aCtx, Errors);

except on E: Exception do
    GetCpErrorsParam(E, Errors, 176, Name, ['ExemploMultiInsert'], CtxSGCA);
end;
finally
    SetComplete;
end;
end;

```

```

FieldsCount(CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var
Errors: ICpErrors): integer

```

Retorna o número de campos da propriedade *Fields* do atributo *MetaComp*. Usado para montar os parâmetros *Values*, que são passados para os métodos *Insert* e *Update*.

Exemplo:

```
Values := VarArrayCreate([0, FieldsCount(CtxSGCA,aCtx,Errors)-1], varVariant);
```

```

LkFieldsCount(CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var
Errors: ICpErrors): Integer

```

Retorna o número de campos da propriedade *LkFields* do atributo *MetaComp*. Usado para montar os parâmetros *LKValues*, que são passados para os métodos *GetRecord* e *Delete*.

```

IndexInFields(const FieldName: WideString; CtxSGCA: OleVariant; const
CtxDb: IBsContextoDb; var Errors: ICpErrors): Integer

```

Retorna o índice de um campo na propriedade *Fields* do atributo *MetaComp*. Usado para montar os parâmetros *Values*, que são passados para os métodos *Insert* e *Update*.

```
IndexInLkFields(const FieldName: WideString; CtxSGCA: OleVariant;
const CtxDb: IBsContextoDb; var Errors: ICpErrors): Integer
```

Retorna o índice de um campo na propriedade *LkFields* do atributo *MetaComp*. Usado para montar os parâmetros LKValues, que são passados para os métodos *GetRecord* e *Delete*.

```
DisplayLabelSingular(CtxSGCA: OleVariant; const CtxDb: IBsContextoDb;
var Errors: ICpErrors): WideString
```

Retorna o valor da propriedade *DisplayLabelSingular* de *MetaComp*. Essa informação é cadastrada no Catálogo de Tabelas.

```
DisplayLabelPlural(CtxSGCA: OleVariant; const CtxDb: IBsContextoDb;
var Errors: ICpErrors): WideString
```

Retorna o valor da propriedade *DisplayLabelPlural* de *MetaComp*. Essa informação é cadastrada no Catálogo de Tabelas.

#### 1.1.2.5 Integração entre Objetos TDmBusiness e Objetos TBsContextoDb

Um servidor *stateless* deve, por definição, alocar os recursos que necessita a cada chamada de método, liberando-os no final da execução do mesmo. Em uma aplicação qualquer que acesse o SGBD, isto significa instanciar uma conexão com o SGBD, fazer as operações necessárias e então fechar a conexão.

Os objetos *TDmBusiness*, entretanto, não fazem acesso direto ao SGBD, mas sim a objetos *TBsContextoDb*. Então, em vez de instanciar uma conexão com o SGBD, eles devem instanciar um destes objetos, fazer solicitações a ele e, quando do final do método, liberá-lo.

Quando se trabalha com objetos COM usando o Delphi, não é necessário escrever código para liberar tais objetos da memória. O compilador Delphi adiciona o código necessário para esta tarefa durante a compilação do programa. Como um objeto *TBsContextoDb* é um objeto COM, então o programador não precisa se preocupar em liberá-lo ao final do método, pois o compilador Delphi irá tratar disto.

Então, nos objetos servidores, em cada método que acesse o SGBD deve ser instaciado um objeto *TBsContextoDb* e logo em seguida chamado seu método *InitContexto* (este processo pode ser feito através de uma chamada à função *ConsisteBsContextoDb*). O objeto instanciado será responsável por fazer a comunicação com o SGBD durante a execução do método.

Abaixo está um exemplo completo de um método criado na servidora *TdmContribuinteImob*:

```

function TdmContribuinteImob.GetProcessosByIdContribuinte(
    IdContribuinte: Integer; CtxSGCA: OleVariant;
    const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant;
var aCtx: IBsContextoDb;
    MetaQr: TQrGetProcessosByIdContribuinte;
begin
try
try
    //Instancia um objeto TBsContextoDb
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    //Cria a MetaQuery
    MetaQr := TQrGetProcessosByIdContribuinte.Create;
    try
        //Configura o parâmetro da query
        MetaQr.ParamByName('ID CONTRIBUINTE').AsInteger := IdContribuinte;
        //Executa a consulta no Banco, e retorna os dados no formato OleVariant, mapeado
        //para ser utilizado em um TClientDataSet
        result := aCtx.OpenBsQuery( MetaQr.PackForCom(CtxSGCA), CtxSGCA, Errors);
    finally
        //Libera a MetaQuery da memória
        MetaQr.Free;
    end;
except
    //Se houve algum erro durante a execução do método, coloca as informações do erro
    //na variável Errors para ser mostrada na aplicação cliente que chamou o método
    on E: Exception do
        GetCpErrorsParam(E, Errors, 176, Name, ['GetProcessosByIdContribuinte'], CtxSGCA);
    end;
finally
    //Indica que o método COM finalizou e todos os objetos podem ser liberados
    SetComplete;
end;
end;

```

#### 1.1.2.6 Controle de Transações nos Objetos Servidores

Como os objetos servidores usam objetos *TBsContextoDb* para fazer acesso ao SGBD, é interessante que eles usem estes mesmos objetos para fazer o controle de transação, uma vez que eles suportam este tipo de operação.

A princípio não existe nenhum problema em usar o suporte a transações fornecido pelos objetos *TBsContextoDb*. Como cada método do objeto servidor que acesse o SGBD instancia um objeto *TBsContextoDb*, basta fazer chamadas aos métodos *InitDbTrans*, *CommitDbTrans* e *RollbackDbTrans* deste objeto. Estes métodos já foram explicados na seção [Contexto](#)<sup>9</sup>.

A idéia acima funciona perfeitamente, mas tem um problema a ser resolvido. Desta maneira não é possível fazer transações entre dois objetos servidores diferentes, pois cada um, ao receber uma chamada, instancia um objeto *TBsContextoDb* para si, o que faz com ambos estejam ocupando diferentes conexões com o SGBD.

Para ilustrar este problema, vai ser considerado um exemplo de um sistema integrado que possui um objeto para fazer operações sobre documentos (*TDocumento*) e um objeto para fazer operações sobre almoxarifados (*TAlmoxarifado*).

Supondo que a cada solicitação de produto feita a um almoxarifado deve ser gerado um documento que registre tal solicitação. Neste caso, o objeto *TAlmoxarifado* poderia iniciar uma

transação (*InitDbTrans*), fazer o processamento necessário e após fazer uma chamada ao método de *TDocumento* que cria um documento. Se tudo corresse bem a transação seria efetivada (*CommitDbTrans*) e caso contrário, desfeita (*RollbackDbTrans*). Entretanto, se o objeto *TDocumento*, ao receber a chamada para criar um documento instanciar seu próprio objeto *TBsContextoDb*, as operações efetuadas por ele não serão desfeitas quando da chamada a *RollbackDbTrans* no código do objeto *TAlmoxarifado*, pois cada um dos objetos estará usando uma instância diferente de *TBsContextoDb*.

A solução mais óbvia para este tipo de problema seria colocar o código responsável por criar um documento dentro do objeto *TAlmoxarifado*, mas esta solução simplesmente elimina o reaproveitamento de código e nem deve ser considerada.

Diante deste problema, o Framework propõem uma segunda solução que é mais eficiente.

A solução proposta e implementada pelo Framework utiliza o controle de transações dos objetos *TBsContextoDb*. No início desta seção foi levantado o problema de se instanciar um objeto *TBsContextoDb* em cada chamada de método. Esta prática torna impossível o uso de transações entre dois objetos diferentes, mas é necessária para que os objetos sejam *stateless*.

A solução do Framework para este problema é simples. Cada método de um objeto *TDmBusiness* deve receber como parâmetro um ponteiro para um objeto *TBsContextoDb*. Assim, no início da execução de cada método, antes de instanciar um objeto *TBsContextoDb*, o programador verifica se o ponteiro para *TBsContextoDb* é válido. Se for válido, ele o usa para fazer acesso ao SGBD, se não for, ele instancia um novo objeto (esta verificação da existência de um objeto *TBsContextoDb* também é resolvida pela chamada à função *ConsisteBsContextoDb*).

Com o mecanismo de compilação do Delphi, que adiciona código para liberar objetos COM, o objeto *TBsContextoDb* passado como parâmetro é liberado quando do término do método onde ele foi instanciado. Ao ser liberado, o objeto *TBsContextoDb* libera a conexão com o SGBD que estava utilizando para que outros objetos possam utilizá-la.

Foi citado anteriormente que o modelo de programação proposto pelo Framework levava a um aproveitamento ótimo do uso das conexões disponibilizadas pelos objetos *TBsContextoDb*. É através do modelo de programação proposto acima e do mecanismo de compilação do Delphi que isto é conseguido, pois com ele os objetos *TBsContextoDb* só permanecem ocupando conexões durante o tempo de execução do método em que foram instanciados, o que é exatamente o tempo mínimo possível durante o qual a conexão deve ser ocupada.

Além disso, como o objeto *TBsContextoDb* suporta transações encadeadas mediante sucessivas chamadas a *InitDbTrans*, é possível que dois ou mais métodos que possuem em sua implementação uma chamada para *InitDbTrans* sejam executados normalmente utilizando o mesmo objeto *TBsContextoDb*. Se qualquer um dos métodos falhar e fizer uma chamada a *RollbackDbTrans*, a transação será desfeita, mesmo se outros métodos que usam o mesmo objeto *TBsContextoDb* fizerem chamadas a *CommitDbTrans*. Este encadeamento de transações possibilita um poder de programação bastante grande.

Abaixo está um exemplo de uso de transação em um método servidor:

```

procedure TdmContribuinteImob.Delete(LKValues: OleVariant;
  Concorrencia: Integer; CtxSGCA: OleVariant; const CtxDb: IBsContextoDb;
  var Errors: ICpErrors);
var
  aCtx: IBsContextoDb;
  aEndereco: IEnderecoDisp;
begin
  try try
    //Instancia um objeto TBsContextoDB
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    //Busca referência para o objeto TdmEndereco, para utilizar os métodos
    //publicados na interface
    aEndereco := GetBsComObj(CLASS_Endereco) as IEnderecoDisp;
    //Inicia a transação no SGBD
    aCtx.InitDbTrans(CtxSGCA, Errors);
    //Obrigatoriamente após abrir a transação, deve existir um bloco try .. except
    try
      //chama o método DeleteEnderecosPorOrigem da interface IEndereco para apagar
      //o endereço do contribuinte
      aEndereco.DeleteEnderecosPorOrigem(Integer(LKValues[0]), 1, CtxSGCA, aCtx, Errors);
      //Se não houve erros, apaga o contribuinte do SGBD
      if SemErro(Errors) then
        inherited Delete(LKValues, Concorrencia, CtxSGCA, aCtx, Errors);
      //Se não houve erros nos dois métodos, efetiva a transação (Commit), caso
      //contrário, desfaz (RollBack)
      if SemErro(Errors) then
        aCtx.CommitDbTrans(CtxSGCA, Errors)
      else
        aCtx.RollBackDbTrans(CtxSGCA, Errors);
      //Final do bloco da transação, qualquer exceção desfaz a transação (RollBack)
    except
      aCtx.RollBackDbTrans(CtxSGCA, Errors);
      raise;
    end;
  except
    on E:exception do
      GetCpErrorsParam(E, Errors, 176, name, ['Delete'], CtxSGCA);
  end;
  finally
    SetComplete;
  end;
end;

```

#### 1.1.2.7 Manipulação de Erros

Qualquer exceção ocorrida durante a execução de um método servidor, se não tratada dentro do mesmo, será redirecionada para a aplicação cliente. Até aí tudo bem, o problema é que esta exceção chega na aplicação cliente como uma exceção da classe *EOleSysError*. Se a exceção havia sido gerada por um erro durante o acesso ao SGBD, a aplicação cliente não terá acesso, por exemplo ao código de erro que foi gerado, o que pode ser uma informação importante para o DBA (*Database Administrator* ou Administrador do SGBD) da empresa.

Assim, o máximo que uma aplicação cliente consegue saber a respeito de uma exceção é sua mensagem e se ela foi gerada durante a execução de seu próprio código ou durante a execução do código da aplicação servidora. Isto sem considerar que o erro pode ter ocorrido durante a chamada para a aplicação servidora. Além disso, apesar de saber que o erro ocorreu na aplicação servidora a aplicação cliente não sabe exatamente em qual das aplicações servidoras ocorreu o erro e em que momento ele ocorreu.

Para solucionar estes problemas, o Framework define uma interface chamada *ICpErrors*, que é implementada pelo objeto COM *TCpErrors*. Esta interface contém informações a respeito de não apenas um erro, mas sim uma lista de erros, de maneira que erros que ocorram em diferentes aplicações servidoras possam ser "empilhados" e disponibilizados, um a um para a aplicação cliente.

Assim, para cada método de qualquer aplicação servidora é passado por referência um ponteiro para um objeto *TCpErrors*. Cada método em uma aplicação servidora deve ter todo o seu código escrito dentro de um bloco *try-except*. Desta maneira, qualquer exceção pode ser tratada no bloco *except-end* e configurada de maneira apropriada para ser adicionada no objeto *TCpErrors*.

O método que adiciona um erro a um objeto *TCpErrors* é o método *Add*. Ele recebe como parâmetro um valor do tipo *TErrorData*, que é um alias para o tipo OleVariant. O número de erros em *TCpErrors* é dado pelo método *Count* e o método que recupera um erro de um *TCpErrors* é *Items*, que recebe um integer como parâmetro e devolve um valor no formato *TErrorData*. O tipo *TErrorData* é, na verdade, uma estrutura com o seguinte formato:

**ExceptionClass**: classe da exceção que foi gerada.

**MsgUser**: mensagem a ser mostrada ao usuário.

**ErrorCode**: código de erro gerado pelo BDE.

**NativeError**: código de erro gerado pelo SGBD.

**ExceptKind**: tipo da execução (pode assumir os valores *tekException*, *tekWarning* ou *tekInformation*).

**AppServer**: nome da aplicação servidora onde foi gerada exceção.

**MsgException**: mensagem contida na exceção.

**FieldName**: nome do campo que gerou o erro (no caso do erro ter sido gerado durante a consistência de algum valor a ser inserido/alterado na tabela).

Assim como existem classes que abstraem do programador o protocolo de comunicação com os objetos *TBsContextoDb*, a classe *TCompError* abstrai o protocolo de comunicação com os objetos *TCpErrors*. A classe *TCompError* possui um atributo para cada posição da estrutura *TErrorData*. São eles: *ExceptionClass*, *MsgUser*, *ErrorCode*, *NativeError*, *ExceptKind*, *AppServer*, *MsgException* e *FieldName*.

Além destes atributos, ela possui um atributo *Data* que contém os valores dos atributos acima, empacotados no formato *TErrorData*. Qualquer alteração no valor da propriedade *Data* é imediatamente refletida nas propriedades acima, assim como uma alteração em qualquer uma das propriedades acima também altera o valor da propriedade *Data*.

A classe *TCompError* pode então ser usada tanto nas aplicações cliente quanto nas aplicações servidoras. Nas aplicações servidoras ela é usada para facilitar a inserção de um erro no objeto *TCpErrors*, e nas aplicações cliente para facilitar a interpretação de um erro que esteja

dentro deste objeto.

Além da classe *TCompError*, existem ainda duas funções disponibilizadas pelo Framework para facilitar ainda mais o tratamento de exceções. Uma delas para ser usada nas aplicações cliente (*TrataCpErrors*) e outras duas para ser usadas nas aplicações servidoras (*GetCpErrors* e *GetCpErrorsParam*).

#### 1.1.2.7.1 Principais Métodos

```
GetCpErrors(Exc: Exception; var Errors: ICpErrors; IdMsg: Integer;  
AppServer: string);
```

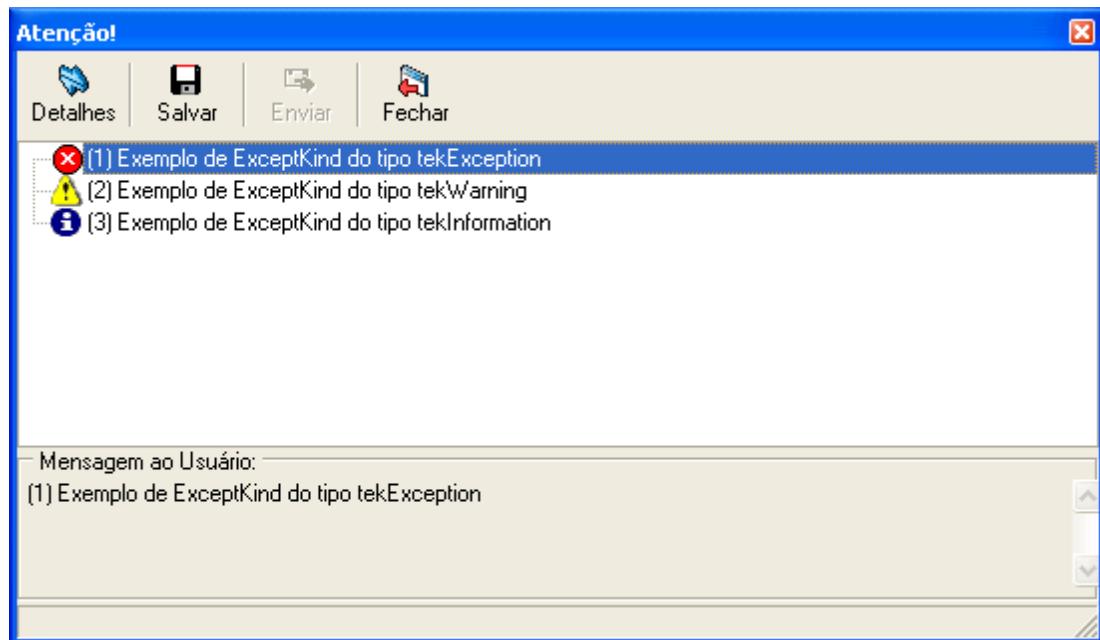
Esta função abstrai ainda mais do programador o modo como são empacotadas as exceções. Ela recebe como parâmetro uma exceção (*Exc*), um ponteiro para um objeto *TCpErrors* (*Errors*), uma mensagem destinada ao usuário (*IdMsg*) e o nome da aplicação servidora (*AppServer*). O que esta função faz é retirar as informações interessantes disponíveis na variável *Exc* (como *ErrorCode*, *NativeError*, *ExceptionClass* e *FieldName*) e adicioná-las, juntamente com os parâmetros *Msg* e *AppServer* na ponteiro *CpErrors*. Se o ponteiro está inválido, a função instancia um objeto *TCpErrors* e coloca o seu endereço nele.

```
GetCpErrorsParam(Exc: Exception; var CpErrors: ICpErrors; IdMsg:  
Integer; AppServer: string; Params: array of const);
```

Semelhante à função *GetCpErrors*. A diferença entre as duas é o parâmetro *Params*, que recebe valores para parâmetros que existam e devam ser substituídos na mensagem referente à *IdMsg*.

```
TrataCpErrors(Errors: ICpErrors);
```

Exibe uma caixa de mensagem se houver algum erro na variável *Errors*. Abaixo segue a tela padrão exibida, caso exista alguma mensagem dentro da variável *Errors*:



```
SemErro(const Errors: CpErrors): Boolean
```

Retorna True se não houver nenhum erro e False se houver algum erro. Somente é considerado erro se houver algum *ExceptionKind* do tipo *tekException*. Na tela exemplificada acima, se houvesse somente as duas últimas mensagens dentro de Errors, o método *SemErro* retornaria False.

Do ponto de vista do programador, então, o tratamento de erro na parte servidora é feito através da declaração de um parâmetro do tipo *ICpErrors* com o modificador *var* em cada método, da proteção do código do método servidor dentro de um bloco *try-except* e da chamada a *GetCpErrors* dentro do bloco *except-end*. E na parte cliente, através da chamada a *TrataCpErrors* após cada chamada a um método servidor. Exemplo do tratamento de erro em um método servidor:

```
function TdmCadastroEconomico.GetContadorByIdCadEconomico( IdCadEconomico: Integer;
    CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant;
var aCtx: IBsContextoDb;
    MetaQr: TQrGetContadorByIdCadEconomico;
begin
  try try
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    MetaQr := TQrGetContadorByIdCadEconomico.Create;
    try
      MetaQr.ParamByName('ID_CAD_ECONOMICO').Value := IdCadEconomico;
      result := aCtx.OpenBsQuery( MetaQr.PackForCom(CtxSGCA), CtxSGCA, Errors);
    finally
      MetaQr.Free;
    end;
  except
    on E: Exception do
      GetCpErrorsParam(E, Errors, 176, Name, ['GetContadorByIdCadEconomico'], CtxSGCA);
  end;
  finally
    SetComplete;
  end;
end;
```

Exemplo da chamada do método e tratamento de erro na aplicação cliente:

```
function TCliCadastroEconomico.GetContadorByIdCadEconomico( IdCadEconomico: Integer): Boolean;
var Errors: ICpErrors;
    Data: OleVariant;
begin
  cdsGetContadorByIdCadEconomico.Close;
  Data := (RemoteBusiness as ICadastroEconomicoDisp).GetContadorByIdCadEconomico(
    IdCadEconomico, GetCtxSGCA, nil, Errors);
  result := SemErro(Errors);
  if result then
    cdsGetContadorByIdCadEconomico.Data := Data;
  TrataCpErrors(Errors);
end;
```

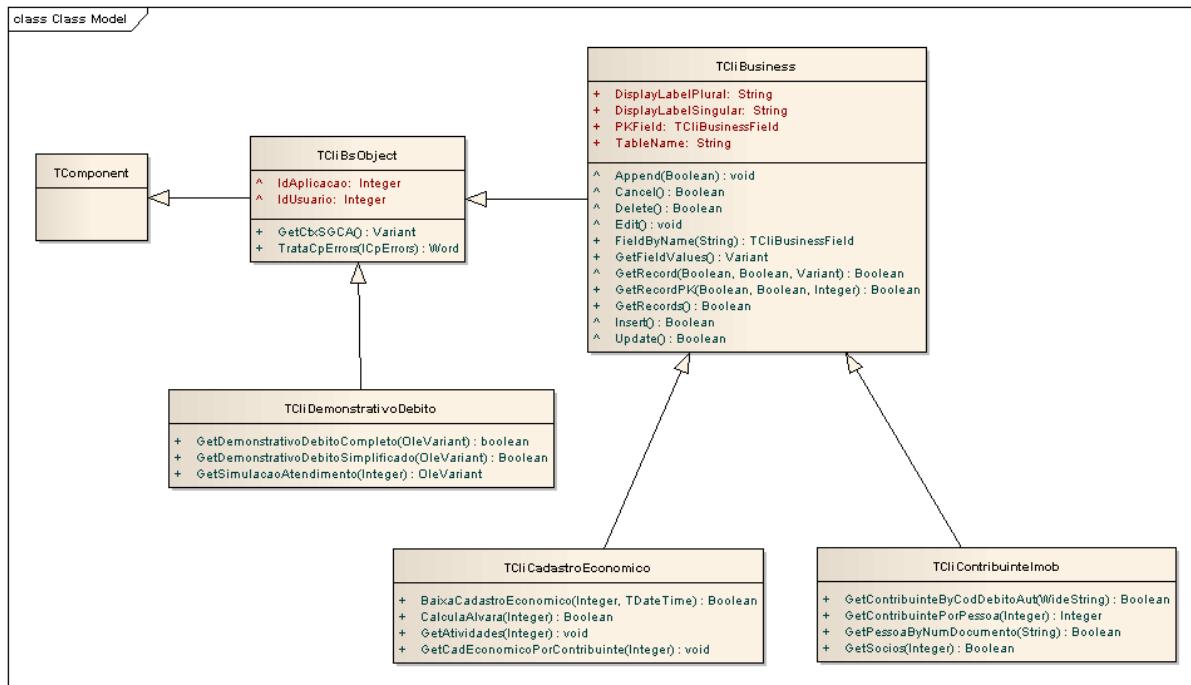
### 1.1.3 Catálogo

O modelo de desenvolvimento contempla a utilização de um Catálogo de Tabelas e Componentes. O Catálogo consiste em um subsistema que tem por objetivo armazenar os metadados das tabelas que compõem o sistema, funcionando como uma espécie de dicionário de dados, ou seja, um mapeamento da estrutura física do banco de dados. É no Catálogo que são armazenados os termos que serão utilizados para cada coluna de cada tabela, qual é a chave física de cada tabela (se existente), rótulos das tabelas e colunas, quais colunas compõem a chave lógica, a obrigatoriedade ou não da entrada de dados para cada coluna, etc. O catálogo possui também informações críticas que são utilizadas internamente pelos Sistemas, de forma que a modificação de suas configurações deve ser efetuada com cautela.

Os componentes *TCliBusiness* bem como as aplicações servidoras que herdam de *ICpBusiness* são configurados dinamicamente de acordo com as informações deste catálogo.

### 1.1.4 *TCliBusiness*

Para acessar um servidor COM usamos uma *Type Library*. No SI\*, todo o acesso aos componentes MTS/COM+ é feito por Type Libraries. Para facilitar o acesso a essas bibliotecas foram criados *Wrappers* (os componentes *Cli's*) que encapsulam em componentes Delphi as chamadas aos servidores COM. Todos esses componentes *wrappers* descendem da classe *TCliBusiness*, para as interfaces servidoras que herdam de *ICpBusiness*, ou da classe *TCliBsObject*, para as demais, que definem o comportamento básico para todos os componentes *Cli's*. Na imagem abaixo está a hierarquia das classes *TCliBusiness*.



As aplicações clientes não se comunicam diretamente com os objetos COM servidores, mas sim com classes que são uma espécie de *wrapper* para estes objetos. Estas classes são descendentes da classe *TCliBusiness*, que por sua vez são descendentes da classe *TCliBsObject*, que descende de *TComponent* da VCL do Delphi. Estas classes são chamadas pelo Framework de *ComponentesCli*.

Como no Framework as aplicações clientes devem comunicar-se com os *ComponentesCli* e não com os objetos COM, então para cada objeto COM servidor deve haver um *ComponeteCli* correspondente. Os métodos destes componentes são exatamente os métodos que existem no objeto COM a que se referem. E o código escrito nestes componentes é o código necessário e suficiente para fazer chamadas aos objetos COM e disponibilizar os dados fornecidos por eles como resposta de uma maneira amigável ao programador da aplicação cliente.

A classe *TCliBusiness* faz o mapeamento dos métodos que são definidos na Interface *ICpBusiness*, como por exemplo os método *Insert*, *Update*, *Delete*, etc. Assim, sub-classes de *TCliBusiness* devem mapear somente os métodos definidos na Interface implementada pelo objeto COM a que se referem. No exemplo da figura acima, *TcliContribuinteImob* já possui pelo conceito de herança os métodos *Insert*, *Update* e *Delete*, porém ele possui ainda métodos específicos da regra de negócio de Contribuintes, como por exemplo o método *GetSocios*.

Como os componentes *CliBusiness* funcionam como *wrappers* para os objetos *TDmBusiness*, é necessário que dentro de cada um destes componentes exista uma referência para o objeto COM que está sendo mapeado por ele. Assim, quando da chamada de um método do componente que resultará em uma chamada remota ao objeto mapeado, o componente possa usar esta referência para resolvê-la.

Existem duas maneiras de manter esta referência no componente. Uma é, quando da criação do *ComponeteCli*, criar também o objeto COM por ele mapeado e guardar uma referência para este objeto em uma variável privada. A outra é manter informações no *ComponeteCli* que possibilitem a criação do objeto COM sempre que for necessário fazer uma chamada remota a ele. A primeira alternativa é mais otimizada, entretanto se o servidor onde está o objeto COM sair do ar por alguns instantes, a referência que o *ComponeteCli* mantinha para o objeto se tornará inválida, o que causará um erro. A segunda alternativa soluciona este problema, pois sempre que for necessário fazer uma chamada remota ao objeto COM, um novo objeto será instanciado. Isto, entretanto, faz com que o sistema perca em performance, pois criar um novo objeto COM e então chamar um de seus métodos despende mais tempo do que simplesmente fazer uma chamada a um objeto já existente. O Framework está implementado utilizando-se da segunda alternativa.

Quando um *ComponeteCli* é instaciado, o primeiro código executado por ele é uma chamada ao seu objeto COM correspondente solicitando as configurações dos seus campos. Isto é feito através de uma chamada ao método *GetFieldsConfig*, que está definido na Interface *ICpBusiness*. A configuração dos campos é armazenada nos *ComponentesCli* de maneira semelhante à usada na classe *TMetaComponent*. Existe uma propriedade *Fields*, que é da classe *TcliBusinessFields* que contém uma coleção de objetos da classe *TcliBusinessField*, sendo que cada um destes objetos representa um campo do componente. A principal diferença dos *ComponentesCli* para a classe *TMetaComponent* quanto ao modo de armazenar os campos é

que esta apenas guarda meta-dados a respeito de seus campos, enquanto que os *ComponentesCli* podem conter valores para estes campos. Os valores para os campos são armazenados na propriedade *CliDsGetRecord*, que é da classe *TClientDataset* e contém os dados retornados pela chamada ao método *GetRecord* (o método *GetRecordPk* também preenche os dados da propriedade *CliDsGetRecord*. Na verdade, as propriedades *CliDsGetRecord* e *CliDsGetRecordPk* fazem referência a uma única instância *TClientDataset*, o que faz com que ambos possam ser utilizados da mesma forma).

Os principais métodos implementados pela classe *TCliBusiness* são os métodos *GetRecord*, *GetRecordPk*, *Update*, *Insert*, *GetRecords*, *GetRecordsLike*, *Append*, *Edit* e *Cancel*.

Os seis primeiros são *wrappers* para métodos homônimos definidos e implementados nos objetos servidores, enquanto os três últimos agem sobre dados locais e não fazem nenhuma chamada remota.

O método *GetRecord* interpreta os dados retornados pelo método homônimo do objeto servidor e preenche os campos do *ComponenteCli* com os valores retornados. Estes valores podem ser editados pelo programador através de uma chamada ao método *Edit*. Para salvar as modificações feitas nos valores dos campos, é usado o método *Update*, que empacota os dados que estão nos campos no formato OleVariant e os passa como parâmetro para o método *Update* do objeto servidor. O procedimento para inserir um novo registro no SGBD é semelhante, entretanto os métodos usados para isto são *Append*, que limpa os valores dos campos, e *Insert*, que empacota os dados no formato OleVariant para passá-los como parâmetro para o método *Insert* do servidor.

Para atribuir valores para os campos de um componente *CliBusiness* é utilizado o método *FieldByName*, que retorna um objeto *TCliBusinessField* que pode ter seu valor alterado através dos métodos *AsString*, *AsDateTime*, *AsFloat*, *AsInteger*, *Value* ou *Clear*.

Exemplo da utilização para *Insert*:

```
procedure TfrmTesteCli.btnTesteInsertCliClick(Sender: TObject);
begin
  CliContribuinteImob.Append(True);
  CliContribuinteImob.FieldByName('NOME CONTRIBUINTE').AsString := 'João da Silva';
  CliContribuinteImob.FieldByName('DT_NASCIMENTO').AsDateTime := StrToDate('25/10/1965');
  CliContribuinteImob.FieldByName('CPF_CNPJ_RG').AsString := '000.000.000-00';
  CliContribuinteImob.FieldByName('NATUREZA_JURIDICA').AsString := 'F';
  CliContribuinteImob.FieldByName('SEXO').AsString := 'M';
  CliContribuinteImob.FieldByName('OBSERVACAO').Value := 'Cadastro de Teste';
  if CliContribuinteImob.Insert then
    ShowMessage('Contribuinte inserido com sucesso!');
end;
```

Exemplo de utilização para *Update*:

```

procedure TfrmTesteCli.btnTesteUpdateCliClick(Sender: TObject);
begin
  //Posiciona o CliDsGetRecord no Contribuinte 13565
  CliContribuinteImob.GetRecordPK(13565, True, True);
  CliContribuinteImob.Edit;
  //Deseja-se somente atualizar a observação do cadastro
  CliContribuinteImob.FieldName('OBSERVACAO').Value := 'Atualização de OBS';
  if CliContribuinteImob.Update then
    ShowMessage('Contribuinte 13565 atualizado com sucesso!!');
end;

```

Os componentes *TCliBusiness* foram desenvolvidos para serem semelhantes aos componentes *TTable*, que faz parte da VCL do Delphi. Estes componentes encapsulam uma tabela, e a alteração e inserção de registros através destes componentes é feita através de chamadas aos métodos *Append* e *Edit* para preparar a edição dos dados seguidas de chamadas ao método *Post* para persistir as alterações efetuadas. Nos componentes *TcliBusiness* a diferença é que em vez do método *Post* são usados os métodos *Insert* e *Update* para persistir os dados. Desta maneira, o programador da aplicação cliente trabalha com os *ComponentesCli* praticamente da mesma forma com que trabalharia com um componente *TTable* quando do seu uso para alteração e inclusão de registros.

O Delphi possui componentes para entrada de dados que interagem perfeitamente com seus componentes de acesso a dados (*TTable*, *TQuery* e *TClientDataset*, que são sub-classes de *Tdataset*). Através de um sistema de "links", é possível associar um componente de entrada de dados (como uma caixa de texto) a qualquer componente que seja sub-classe de *Tdataset*. Uma vez feita esta associação, quando o conteúdo da caixa de texto é alterado, é alterado também o valor do campo a ela associada. O inverso também é verdadeiro, ou seja, ao alterar o valor do campo, o conteúdo da caixa de texto também é alterado.

Para não perder esta e outras facilidades proporcionadas pelos componentes da VCL do Delphi, os componentes *TcliBusiness* usam, internamente componentes da classe *TClientDataset* que, por ser sub-classe de *Tdataset* interage com os componentes de entrada de dados de maneira transparente. Desta maneira, as operações de *Edit*, *Append* e *Cancel* citadas anteriormente são implementadas através de chamadas aos métodos homônimos do componente *TClientDataset*. Como o componente *TClientDataset* interpreta dados que estão no formato OleVariant retornado pelos métodos *OpenBsQuery* do Contexto, quando da chamada remota ao método *GetRecord*, o retorno deste é atribuído ao *TClientDataset* para que ele o interprete. Uma vez feito isto em um *TClientDataset*, ele disponibiliza chamadas a seu método *FieldName* para recuperar o valor de um campo. Assim, o método *FieldName* dos *ComponentesCli* também é implementado através de uma chamada ao seu método homônimo que está disponível no *TClientDataset*. O retorno do método *FieldName* de um *Tdataset* é um objeto da classe *TField*, que disponibiliza os métodos *AsString*, *AsInteger*, *AsFloat*, *AsDateTime*, *Value* e *Clear* para manipular seu valor. Os métodos *AsString*, *AsInteger*, *AsFloat*, *AsDateTime*, *Value* e *Clear* disponíveis na classe *TcliBusinessField* são implementados também através de chamadas a métodos homônimos da classe *TField*.

Com esta implementação, a programação de aplicações clientes com o uso dos

*ComponentesCli* fica bastante semelhante ao modelo de programação tradicional proposto pelo ambiente Delphi, além de possibilitar o uso de todos os componentes de acesso a dados disponibilizados pela VCL do Delphi.

Como os componentes *CliBusiness* são *wrappers* para objetos servidores, a grande maioria dos métodos implementados por eles têm equivalentes no objeto COM que ele está mapeando. Como visto nas Seções anteriores, existem três parâmetros que são comuns a todos os métodos de interface dos objetos servidores:

***CtxSgca***: do tipo *TSgcaPack*, através do qual são enviadas informações do cliente para o servidor; Explicada melhor no [Apêndice A](#)<sup>[146]</sup>

***CtxDb***: do tipo *IBsContextoDb*, Interface usada para fazer acesso ao SGBD. Explicado na seção sobre [controle de transações](#)<sup>[23]</sup>.

***Errors***: do tipo *ICpErrors*, Interface usada para o tratamento de exceções. Explicado na seção de [manipulação de erros](#)<sup>[25]</sup>.

A assinatura dos métodos disponibilizados ao programador da aplicação cliente através dos *ComponentesCli* é exatamente a mesma assinatura dos seus métodos equivalentes nos objetos servidores, com exceção dos três parâmetros citados acima. Estes parâmetros não irão fazer parte da assinatura de nenhum método dos *ComponentesCli*. Assim, o programador da aplicação cliente sequer toma conhecimento da existência da estrutura *TSgcaPack* e de sua importância para o funcionamento do sistema ou da existência de um objeto para fazer acesso a dados (*CtxDb*) ou ainda do modo como é feito o tratamento de erro nas aplicações servidoras. Para ele, um componente *TcliBusiness* é um componente como outro qualquer que possui métodos em sua interface que podem ser utilizados para realizar tarefas.

Basicamente, os métodos implementados nos componentes *TcliBusiness* são de dois tipos: Métodos de Atualização de Dados e Métodos de Recuperação de Dados.

#### 1.1.4.1 Métodos de Atualização de Dados

Um método de atualização de dados é o mais simples de ser implementado. Estes métodos apenas fazem chamadas remotas a métodos, testando o sucesso de sua execução quando de seu retorno. Um método deste tipo tem o seguinte padrão de implementação:

```
function TCliCadastroEconomico.CalculaISSFixo( IdCadEconomico: Integer): Boolean;
var Errors: IcpErrors;
begin
  (RemoteBusiness as ICadastroEconomicoDisp).CalculaISSFixo(IdCadEconomico,
                                                               GetCtxSGCA, nil, Errors);
  result := SemErro(Errors);
  TrataCpErrors(Errors);
end;
```

#### 1.1.4.2 Métodos de Recuperação de Dados

Os métodos de recuperação de dados possuem um código um pouco maior, sua implementação, já que deve interpretar os dados retornados do servidor. Sua implementação,

entretanto também é bastante simples. A principal diferença deste tipo de método para os métodos de atualização de dados é que, para cada método de recuperação deve existir um componente *TClientDataset* equivalente através do qual o programador terá acesso aos dados retornados do servidor. Se em um *ComponenteCli* existir um método *GetSalas* cujo objetivo é retornar um conjunto de salas, nele deve existir também uma propriedade do tipo *TClientDataset* com o nome de *cdsGetSalas* que, quando da chamada do método *GetSalas* receba os dados que retornaram da chamada ao servidor. Assim, o programador, após chamar o método *GetSalas*, vai ter acesso aos dados que retornaram através da propriedade *cdsGetSalas*. Um método de recuperação de dados tem o seguinte padrão de implementação:

```
function TCliCadastroEconomico.GetResponsaveisEcon(
  IdCadEconomico: Integer): Boolean;
var
  Errors: ICpErrors;
  aData: OleVariant;
begin
  aData := (RemoteBusiness as ICadastroEconomicoDisp).GetResponsaveisEcon(IdCadEconomico,
    GetCtxSGCA, nil, Errors);
  Result := SemErro(Errors);
  if Result then
    cdsGetResponsaveisEcon.data := aData;
    TrataCpErrors(Errors);
end;
```

Pode-se notar que, com o padrão de implementação proposto para os *ComponentesCli*, o programador sequer toma conhecimento que existe uma rotina de tratamento de erro e a única resposta que ele recebe da chamada a um destes métodos é se ele executou com sucesso ou não. Entretanto, pode haver casos em que seja necessário ao programador tomar atitudes diferentes dependendo do tipo de erro que ocorreu. Estes são casos específicos e o Framework não propõe nenhum padrão para eles, mas a solução pode ser trocar o retorno do método para um tipo *Integer* ou mesmo definir um parâmetro de saída (modificador *out*) e definir algumas constantes para os erros a serem testados.

### 1.1.5 Componentes Visuais

O Delphi possui componentes de entrada de dados que podem ser associados com componentes de acesso a dados (como os componentes *TTable* e *TClientDataset*, ou qualquer outro que seja sub-classe de *TDataset*). Estes componentes de entrada de dados são chamados de componentes *data-aware*. A cada componente *data-aware* é associado um componente da classe *TDataset* e a partir dele, é possível escolher o campo ao qual ele irá ficar associado.

Os componentes *CliBusiness* interagem com os componentes *data-aware* disponibilizados pela VCL do Delphi através de propriedades da classe *TClientDataset*.

Apesar da interação com estes componentes ser transparente, algumas facilidades disponíveis para o programador da aplicação cliente ficam prejudicadas. No desenvolvimento de uma aplicação tradicional Delphi, o programador pode selecionar um componente *TDataset* que esteja "colado" em seu formulário a partir de uma lista e a partir daí selecionar um de seus campos.

Com o uso dos componentes *TCliBusiness* isto não é possível, pois os componentes

*TClientDataset* não ficam "colados" no formulário, e sim fazem parte dos *TCliBusiness*. Para solucionar este problema foram desenvolvidos componentes *data-aware* a partir dos componentes Delphi. Estes componentes apenas possuem uma propriedade para o programador informar o *TcliBusiness* que deseja usar e, a partir daí informar um dos componentes *TClientDataset* existentes dentro do componente.

#### 1.1.5.1 TBsContainer

O componente visual *TBsContainer* é o componente visual mais completo para manipulação de dados. Ele já possui um componente da VCL do Delphi *TLabel*, que possuirá a descrição do campo. Por padrão esse Label conterá o nome da coluna que está sendo manipulada. Além disso, ele possui um controle para informação e manipulação de dados. Quando o *TBsContainer* é adicionado a algum formulário de alguma aplicação, devem ser configuradas algumas propriedades:

**BsClientData**: Conterá as informações do *CliBusiness* e o *FieldName* a serem utilizados para montar a descrição do *TLabel*.

**BsControlType**: O tipo de controle que será utilizado para manipular a informação do campo da tabela física. Os mais comuns são:

***bstEdit***: componente semelhante ao *TDBEdit* da VCL do Delphi.

***bstEditNormal***: componente semelhante ao *TEdit* da VCL do Delphi.

***bstComboBox***: componente semelhante ao *TDBComboBox* da VCL do Delphi

***bstCurrency***: semelhante do *bstEdit*, porém possui informações adicionais para trabalhar com valores decimais, como separador decimal (vírgula ou ponto) e número de casas decimais a serem mostradas no controle.

***bstDateTime***: componente para trabalhar com valores de data e hora. Usuário pode informar a data através de um calendário, semelhante do componente *TDateTimePicker* da VCL do Delphi.

***bstMemo***: componente semelhante ao *TDBMemo* da VCL do Delphi. Nada mais é do que um *bstEdit* com a possibilidade de inclusão de várias linhas.

***bstTabEstrut***: componente para listar os itens de uma tabela estruturada. Utilizado quando o campo a ser preenchido possuir relacionamento com tabela estruturada.

OBS: cada *BsControlType* possui internamente um *BsClientData*, que possuirá as informações de qual *CliBusiness* e qual campo que será persistido, ou seja, que será gravado no SGBD.

Resumindo: O *BsClientData* do *BsContainer* possuirá as informações para configuração do *TLabel*, e o *BsClientData* do *BsControlType* possuirá as informações de onde será gravado os valores informados no controle.

***Localizar***: Se for informado "True", será adicionado um botão "localizar" no lado direito do controle.

**Novo:** Se for informado "True", será adicionado um botão "novo" no lado direito do controle.

#### 1.1.5.2 TBsQuantumGrid

O componente *TBsQuantumGrid* é o componente mais completo para visualização de dados. Além da visualização, os dados podem ser exportados para o formato Excel, HTML, CSV e TXT, além de poderem ser impressos. Em caso de utilização de agrupamento de colunas, pode ainda ser gerado um gráfico com os dados agrupados.

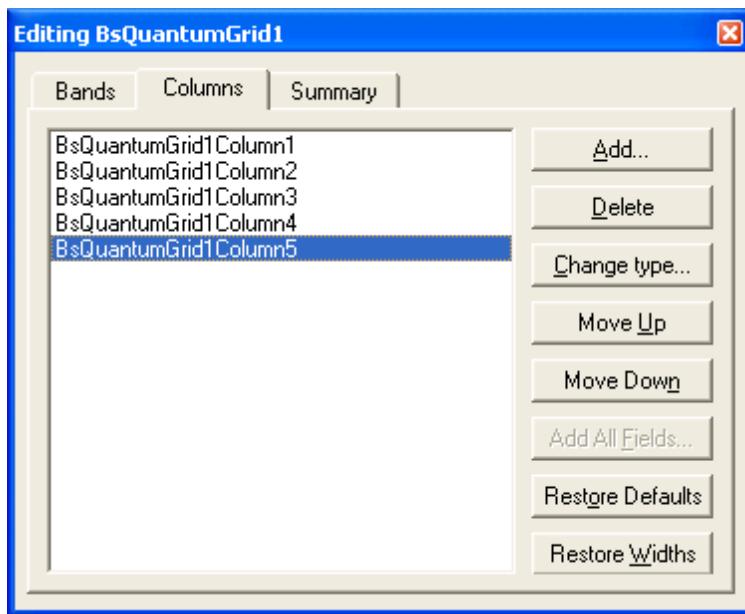
Ele pode ser ligado diretamente aos *TClientDataSet's* dos métodos que retornam dados, dentro do *CliBusiness*, conforme visto na seção sobre "[Métodos de Recuperação de dados](#)"<sup>33</sup>". As principais configurações que devem ser feitas para serem visualizados os dados na *BsQuantumGrid* são:

**BsClient:** Nome do *CliBusiness* de onde está o *TClientDataSet* com os dados a serem mostrados.

**BsDataSource:** Nome do *DataSource* que está vinculado com o *TClientDataSet* com os dados. Geralmente o nome do *DataSource* é "ds" mais o nome do método. No caso do método *GetRecord*, padrão em todos os *CliBusiness*, o nome do *DataSource* é "dsGetRecord", no caso do método "GetSocios", programado para trazer todos os sócios de um contribuinte dentro do *TCliContribuinteImob*, o nome do *TClientDataSet* é "cdsGetSocios" e do *DataSource* "dsGetSocios".

**KeyField:** Cada *TClientDataSet* deve possuir algum campo que identifique o registro como único. Dessa forma é aconselhável que as consultas SQL que trazem dados para visualização, contenham a chave primária de alguma tabela, para ser utilizada nos *KeyFields* dos *BsQuantumGrid*. Se o **KeyField** não for configurado, nada será mostrado na *BsQuantumGrid*, mesmo que o *TClientDataSet* possua dados.

Dando um duplo clique na *BsQuantumGrid* no formulário, será aberta uma tela para ser adicionada as colunas a serem visualizadas:



Para cada coluna adicionada no editor, deverá ser informada a propriedade "*FieldName*", que é o nome do campo a ser visualizado. Obviamente o campo deve estar no *TClientDataSet* que está vinculado com o *BsDataSource* configurado no *BsQuantumGrid*.

### 1.1.6 Componentes de Pesquisa (BsLocalizar)

Além dos componentes *data-aware* para interagir com os componentes *TCliBusiness*, o Framework possui também componentes utilizados para pesquisa de dados. Estes componentes são da classe *TBsLocalizar* e também interagem com os componentes *TCliBusiness*.

A idéia do uso destes componentes é padronizar a maneira como são feitas pesquisas de dados pelo usuário final da aplicação. Cada um destes componentes possui um formulário para pesquisa de dados que é mostrado ao usuário quando da chamada ao seu método *Execute*.

As duas propriedades mais importantes destes componentes são a propriedade *BsClient* e *BsDataSource*. A primeira é da classe *TCliBusiness* e deve ser configurada com o componente *CliBusiness* que implementa a chamada remota ao método que busca os dados para pesquisa. A segunda é o nome do método que efetua a consulta no SGBD. Para buscar os dados solicitados pelo usuário são usadas os recursos de RTTI (Run Time Type Information) fornecidos pelo Delphi, através dos quais é possível fazer uma chamada a um método de um objeto qualquer sabendo apenas seu nome.

Entretanto, não é possível identificar o número de parâmetros que são passados para o método, nem sequer os tipos dos mesmos, a menos que estes sejam na verdade propriedades e não métodos, o que não é o caso dos métodos utilizados pelos componentes *TCliBusiness*.

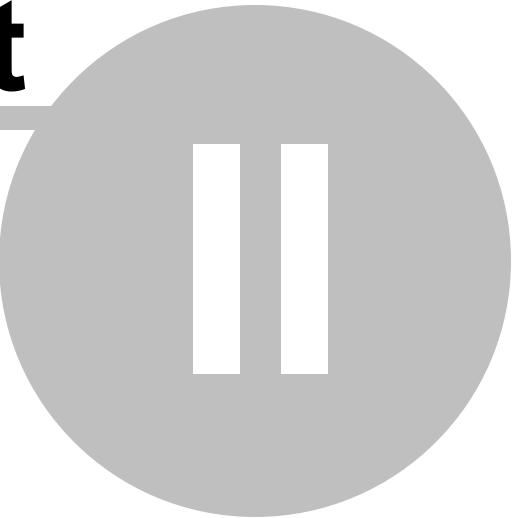
Assim, foi definida uma assinatura de método padrão a qual todos os métodos que poderão ser usados em componentes *TBsLocalizar* devem obedecer. A assinatura destes métodos deve ser a seguinte:

```
Function (Condicao: string; Par1,Par2,Par3,Par4,Par5,Par6: Variant): Boolean;
```

# **Top Level Intro**

This page is printed before a new  
top-level chapter starts

**Part**



II

## 2 Criando uma servidora

Para exemplificar os conceitos abordados neste manual, criaremos uma aplicação nova, desde a parte de banco de dados, aplicação servidora e aplicação cliente.

### 2.1 Tabelas Físicas

Abra o SQLPlus do Oracle ou alguma outra ferramenta que permita a execução de comandos SQL e rode os seguintes scripts de banco:

```
| create table DBSM.PREDIOS_A
| (ID_PREDIO integer not null primary key,
|  ID_BEM integer,
|  NOME_PREDIO varchar(60),
|  LOCALIZACAO varchar(60),
|  OBSERVACAO clob,
|  TIPO_POSSE_ITEM integer,
|  TIPO_POSSE_TAB integer,
|  NUM_PREDIO varchar(10),
|  BLOCO varchar(10),
|  LOCAL_FISICO_TAB integer,
|  LOCAL_FISICO_ITEM integer,
|  COD_OPERADOR integer,
|  DT_ALTERACAO date,
|  HR_ALTERACAO date,
|  CONCORRENCIA integer,
|  ENDERECO_FISICO varchar(15));

create table DBSM.ESPACOS_A
(ID_ESPACO integer not null primary key,
ID_PREDIO integer references PREDIOS_A(ID_PREDIO),
NOME_ESPACO varchar(40),
AREA decimal(11,2),
DIMENSOES varchar(20),
CAPACIDADE integer,
TIPO_ESPACO_ITEM integer,
TIPO_ESPACO_TAB integer,
NUM_SALA varchar(10),
COD_OPERADOR integer,
DT_ALTERACAO date,
HR_ALTERACAO date,
CONCORRENCIA integer,
ENDERECO_FISICO varchar(15));
```

**Nota:** Os campos COD\_OPERADOR, DT\_ALTERACAO, HR\_ALTERACAO, CONCORRENCIA e ENDERECO\_FISICO são obrigatórios em todas as tabelas e contêm informações utilizadas para controle histórico de alteração de registros do banco de dados

### 2.2 Gerador

O modelo de desenvolvimento do SI\* contempla a utilização de um Catálogo, que nada mais é do que um conjunto de tabelas que mapeia a estrutura física do Banco de Dados, contendo informações sobre as tabelas que existem no sistema, bem como suas colunas, chaves primárias,

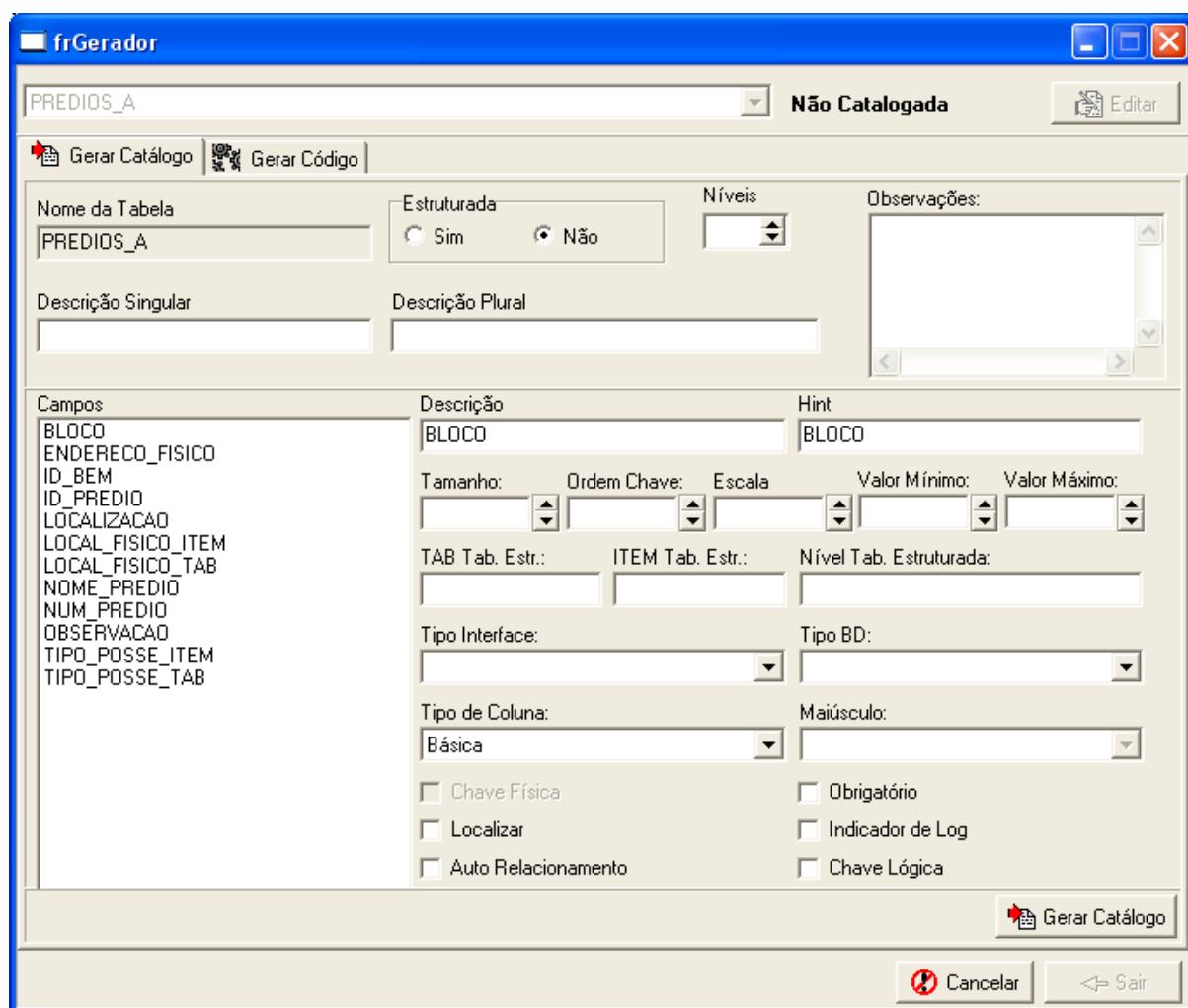
chaves estrangeiras, etc.

Além deste mapeamento, o Catálogo contém informações adicionais que são utilizadas pelo SI\* e que não são contempladas pelo esquema de armazenamento interno do Banco de Dados.

Para agilizar o desenvolvimento de aplicações no SI\* existe um aplicativo chamado Gerador que é responsável por gerar as informações para este Catálogo, bem como criar os arquivos-fonte iniciais referentes à camada servidora das aplicações.

### 2.2.1 Predios A

Execute a aplicação **GeradorEx.exe** (`\|Servidor\AppSI\Gerador\GeradorEx.exe`). Dê o *login* no sistema (obrigatoriamente, o usuário deve ser um "super usuário" para poder abrir a aplicação) e selecionar a tabela *PREDIOS\_A* criada anteriormente. Clique na aba "Gerar Catálogo" e após clique em "Editar".



Preencha os campos Descrição Singular e Descrição Plural como Prédio e Prédios, respectivamente.

Para cada um dos campos adicione as seguintes configurações:

ID_PREDIO	Chave Física = Sim (marcar)  Chave Lógica = Sim  Ordem Chave: 1	Indica que o campo é a primary key da tabela. Normalmente apenas um campo da tabela possui este atributo como true.
NOME_PREDIO	Localizar = Sim  Obrigatório = True	Col. Localizar indica que o campo NOME_PREDIO será utilizado no código gerado como a coluna de parâmetro para o método GetRecordsLike.  Obrigatório indica que o campo NOME_PREDIO deve ter seu valor informado obrigatoriamente, caso contrário um erro será gerado.
TIPO_POSS_E_ITEM	Tab. Estrut. Item = 101	Indica que o campo representa um Item da Tabela Estruturada de número 101.
TIPO_POSS_E_TAB	Tab. Estrut. Tab = 101	Indica que o campo representa o Código da Tabela Estruturada (no caso, 101).
LOCAL_FISICO_ITEM	Tab. Estrut. Item = 216	Indica que o campo representa um Item da Tabela Estruturada de número 216
LOCAL_FISICO_TAB	Tab. Estrut. Tab = 216	Indica que o campo representa o Código da Tabela Estruturada (no caso, 216).

Note que ao configurar um campo como Chave Física, o seu tipo passa a ser *bftAutoInc*.

**Tipo Interface** = tipo de dado definido na Interface ICpBusiness o qual será utilizado pelas aplicações do SI\* para identificar o tipo de cada coluna do Banco de Dados;

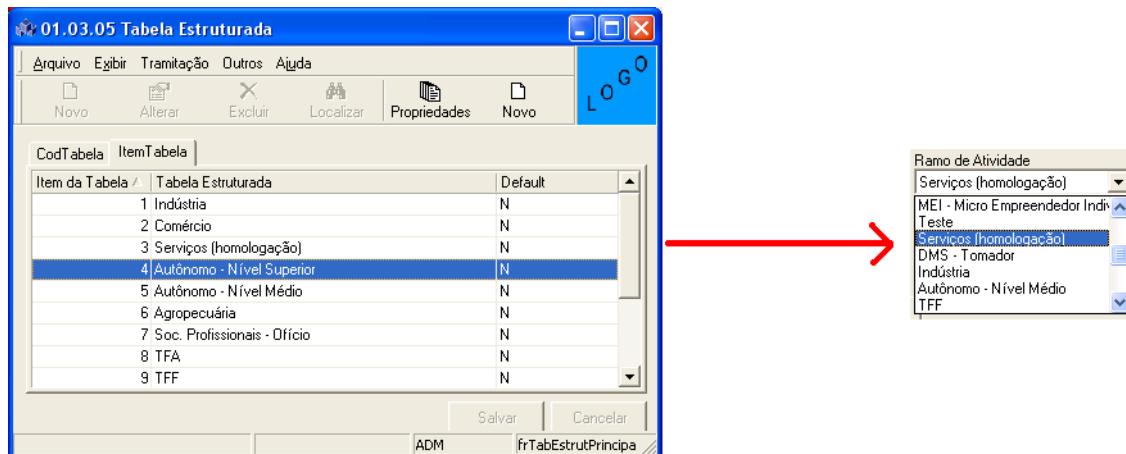
**Localizar** = selecione a coluna que deve ser utilizada como parâmetros das telas de buscas. Este campo será mais tarde utilizado pelo método *GetRecordsLike* para montar a instrução SQL “Like” de forma automática;

**Chave Física** = deve ser somente uma por tabela. O campo marcado com esta opção será utilizado para montagem da instrução SQL do método *GetRecordsPK*;

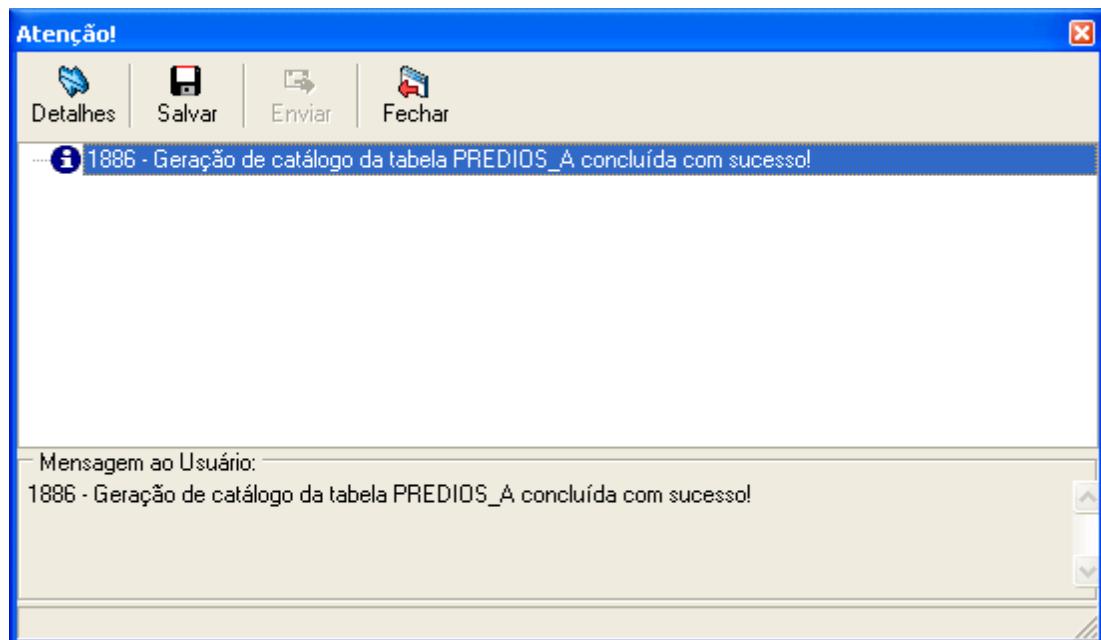
**Chave Lógica** = pode existir mais de uma por tabela e é utilizada pelo método *GetRecord* e pelo método *Delete*;

**TAB Tab. Estru. e ITEM Tab. Estru.**: É uma tabela do banco de dados que armazena uma lista possíveis valores para um determinado campo de outra tabela. A referência a esse item de uma tabela estruturada é feita por dois campos (ITEM e TAB). Isso reduz a necessidade da criação de inúmeras tabelas auxiliares que funcionam como “lookups” para campos-chave de tabelas

relacionadas. Cada tabela estruturada será gravada na tabela física TAB\_ESTRUTURADA, e cada tabela possuirá um número (código), que serão utilizados nos campos TAB e ITEM. Abaixo segue o exemplo da tabela estruturada 224, utilizada no "combo box" da aplicação de cadastro mobiliário:

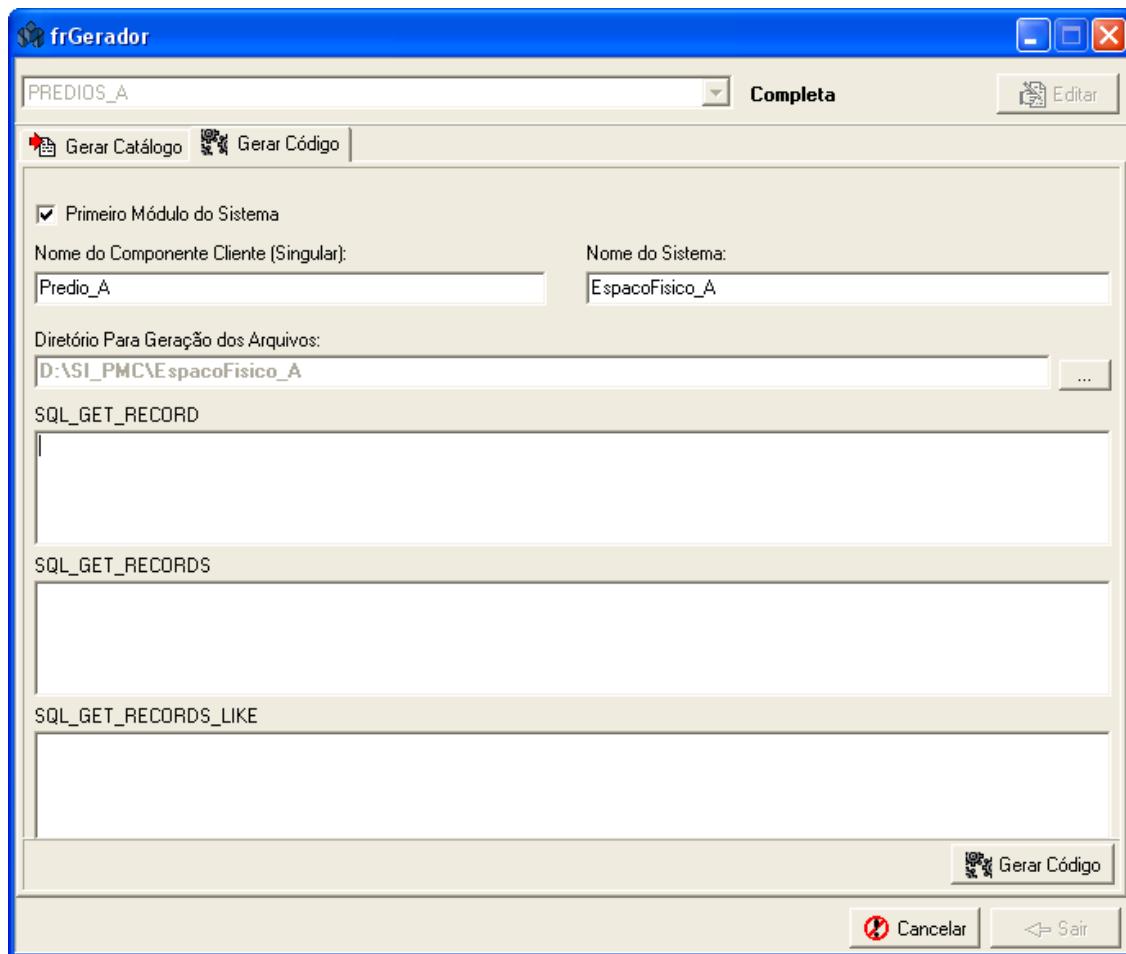


Após efetuar todas as configurações clique no botão "Gerar Catálogo". A aplicação irá gerar o catálogo de tabelas e de componentes e a seguinte tela deverá aparecer:



Após a geração do catálogo iremos criar uma aplicação servidora (DLL COM+) e um pacote de componentes com um componente cliente para esta tabela (herdado de *TCliBusiness*).

Vá na aba *Gerar Código* e preencha as informações como mostra a figura a seguir:



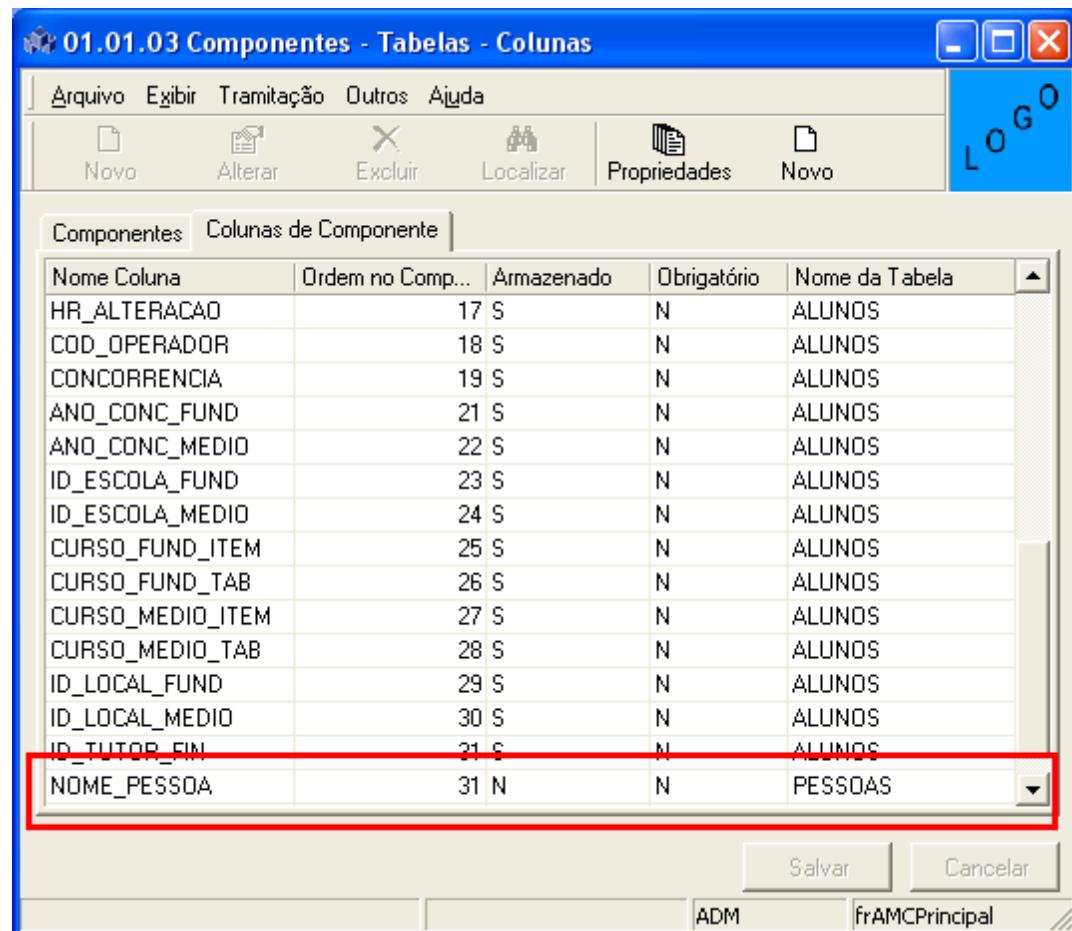
Como é uma servidora nova que estamos criando, marque a opção "Primeiro Módulo do Sistema".

Os campos *SQL\_GET\_RECORD*, *SQL\_GET\_RECORDS* e *SQL\_GET\_RECORDS\_LIKE* são necessários apenas quando a tabela possua "campos não armazenados"

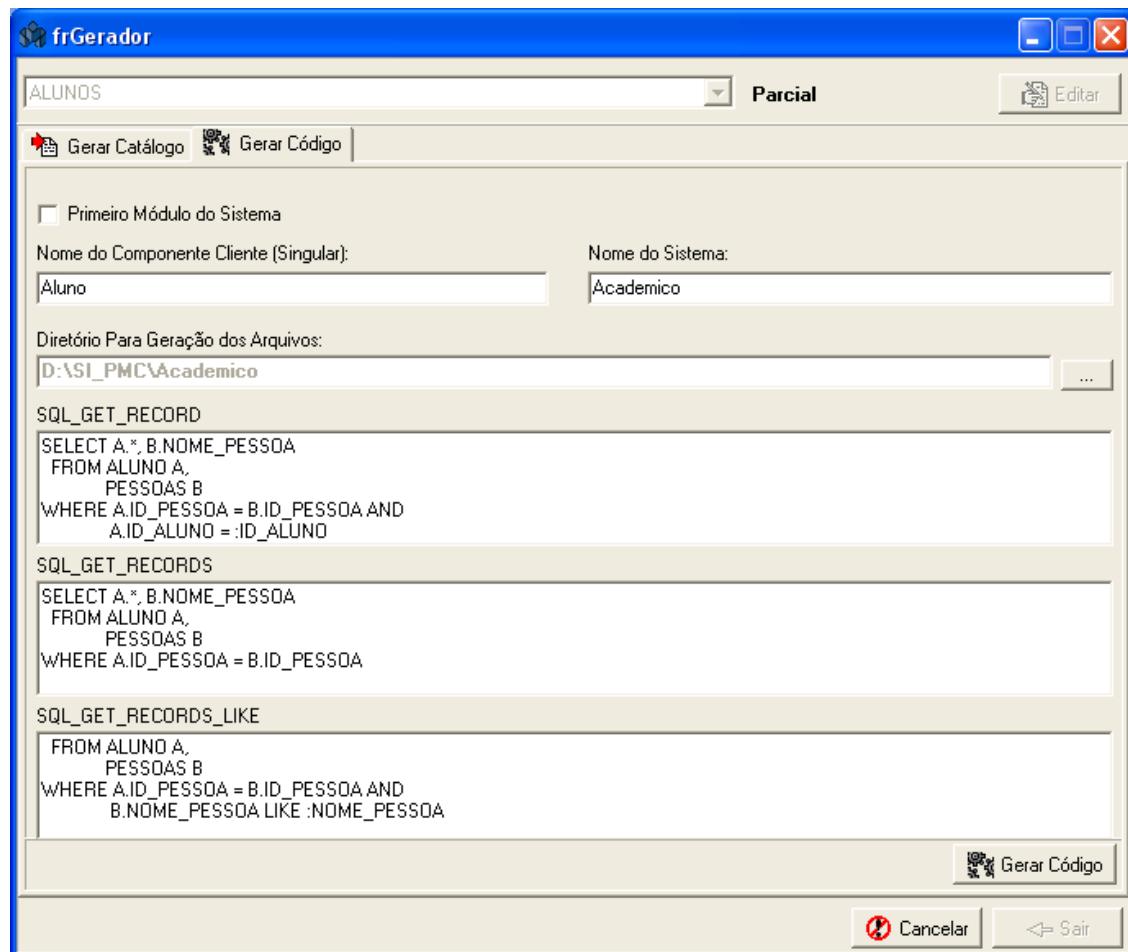
Campos não armazenados são campos provenientes de outras tabelas que são necessários a manutenção desta.

Exemplo: A tabela alunos não possue um campo para armazenar o nome do aluno e sim um campo ID\_PESSOA relacionando um registro em PESSOAS. Levando em consideração que este campo será exibido em grande parte das aplicações este campo é cadastrado no catálogo como um campo não armazenado (apenas no cadastro de componentes, não no de tabela). Como ele deve ser retornado nas consultas comuns do sistema são adicionados estes SQL's com seus respectivos joins para retornar este valor.

Veja como o componente é configurado com um campo não armazenado:



E como o gerador de código deveria ser configurado nesse caso:



Clique no botão "Gerar Código". Após esta operação, serão criadas duas novas pastas dentro do diretório selecionado.

prjNomedoProjeto – contém arquivos da aplicação servidora (DLL) com a implementação dos métodos definidos na interface da Type Library

pknNomedoProjeto – é um pacote Delphi (.DPK) que contém componentes “Cl”, que são *wrappers* para acessar aos servidores PRJ’s, conforme visto na seção [TCliBusiness](#)<sup>[29]</sup>.

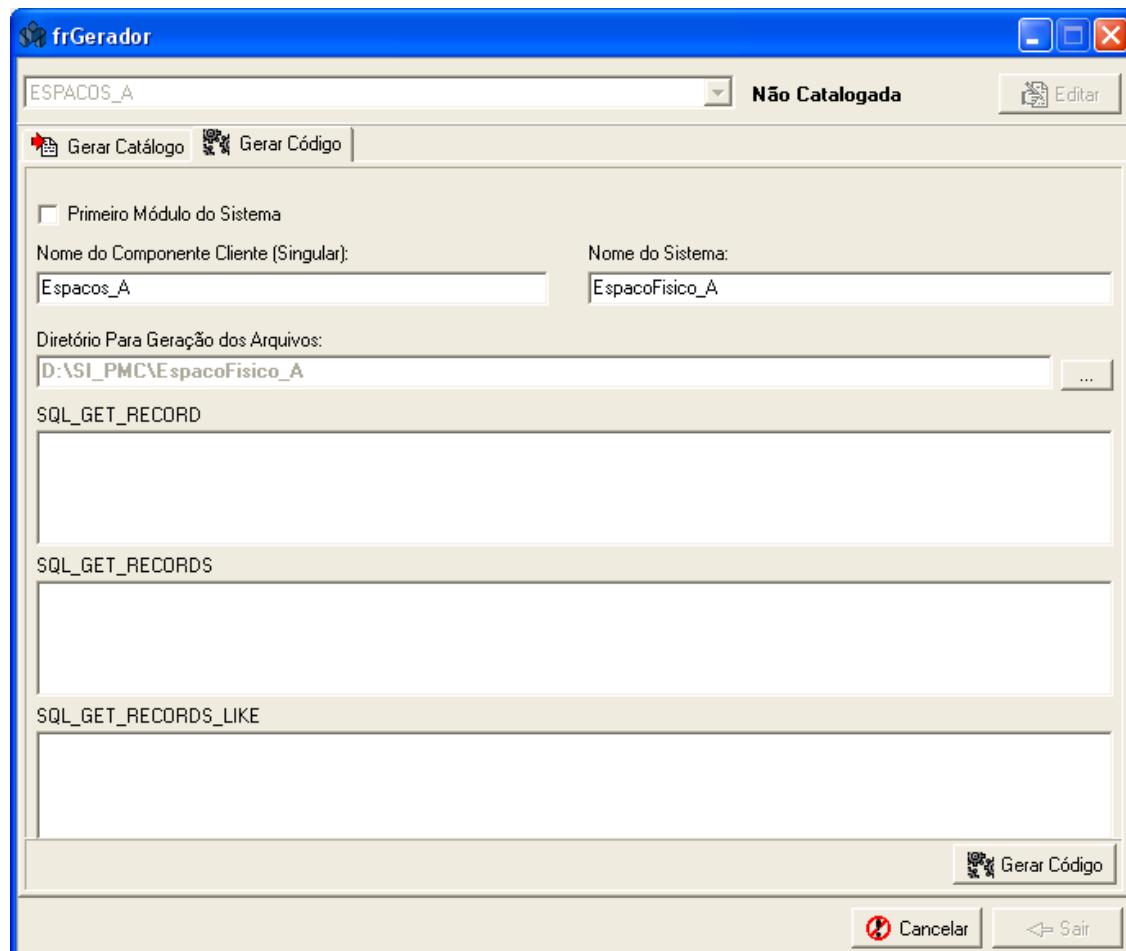
## 2.2.2 Espacos A

Volte para a aba *Gerar Catálogo*, selecione a tabela *ESPAÇOS\_A* e entre com as informações:

ID_ESPACO	Chave Física = Sim Chave Lógica = Sim Ordem Chave: 1
-----------	--

NOME_ESPACO	Localizar = Sim Obrigatório = Sim
TIPO_ESPACO_ITEM	Tab. Estrut. Item = 102
TIPO_ESPACO_TAB	Tab. Estrut. Tab = 102

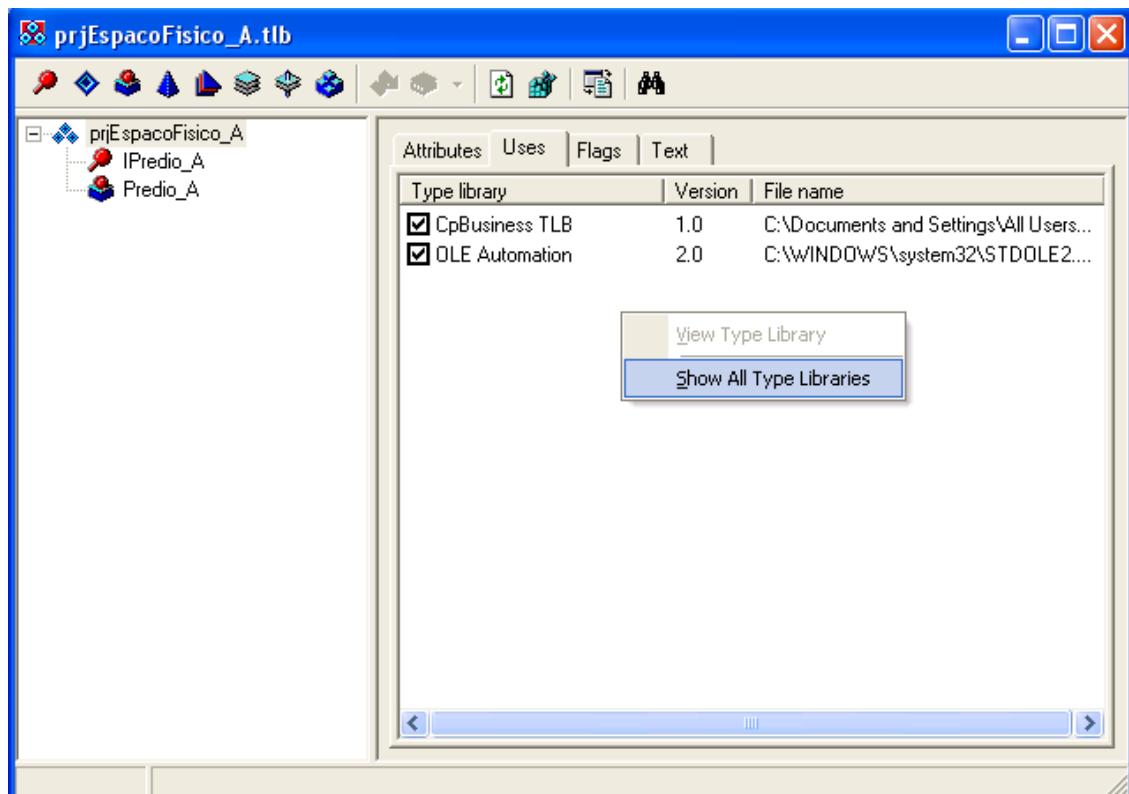
Clique em gerar catálogo e após a geração vá para a aba "Gerar Código" e entre com os dados:



OBS: Já foi criada uma servidora e um pacote de componentes (quando foi criada a Predios A), portanto deixe desmarcada a opção "Primeiro Módulo do Sistema", assim o gerador irá apenas adicionar um novo componente *TClXXX* e uma nova interface na servidora aos pacotes/projetos já existentes.

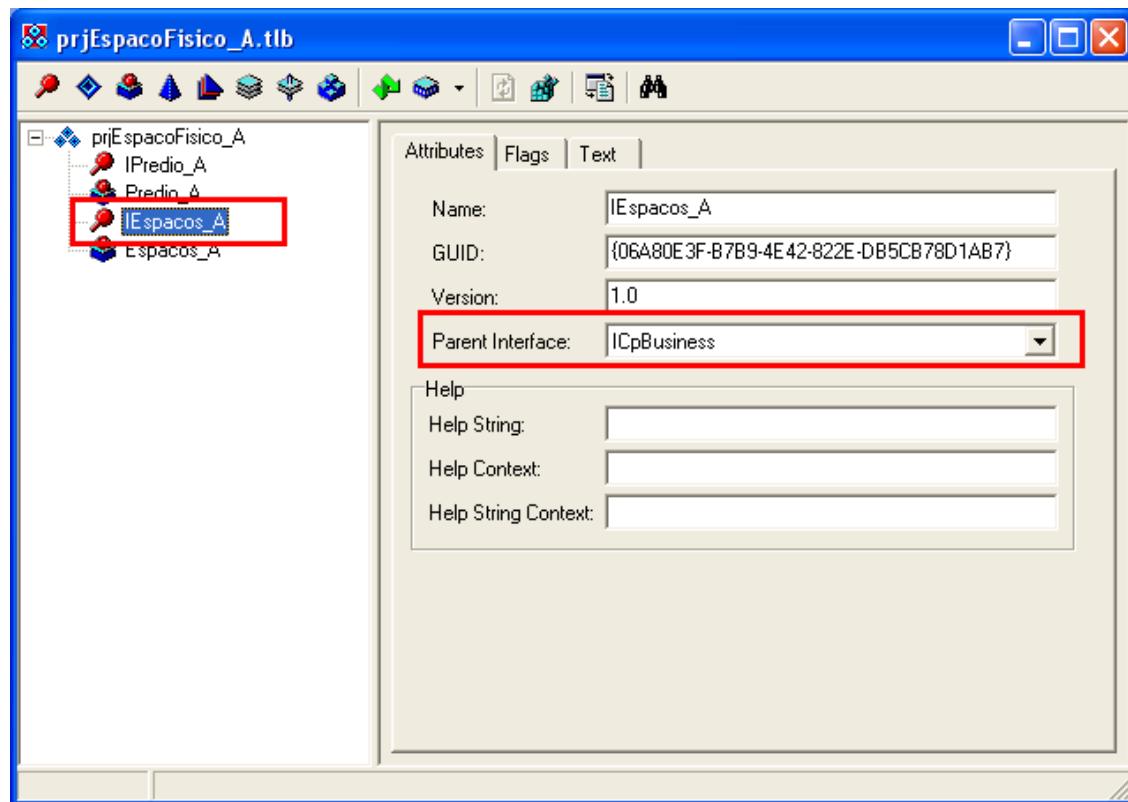
## 2.3 Configurando o Servidor e a Type Library

Abra o arquivo “prjEspacoFisico\_A.dpr” no Delphi 7 dentro do diretório *prjEspacoFisico\_A\Fontes*. Clique então no menu *View | Type Library*.



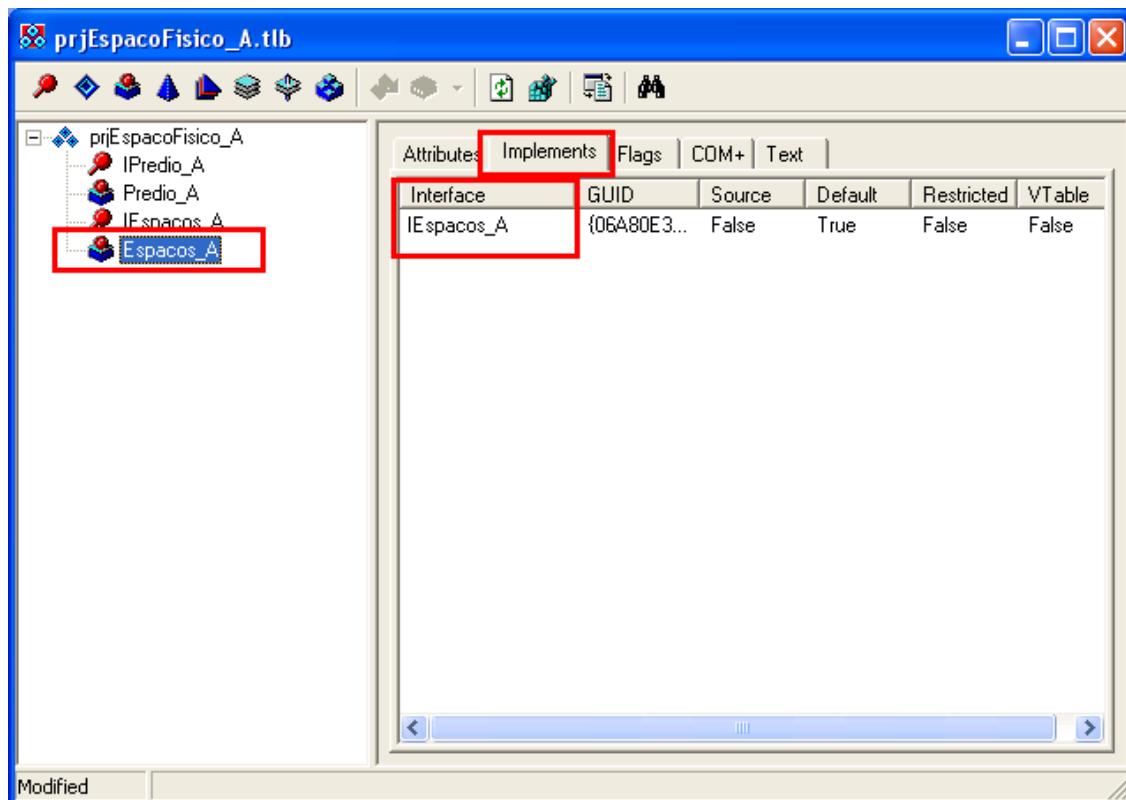
Clique na aba "Uses", senão aparecer todas as *Type Library*, clique com o botão direito e selecione a opção "*Show All Type Libraries*". Marque então as seguintes type libraries: *CpBusiness*, *CpErrors* e *PrjContextoDB*.

O gerador configura a *Type Library* para o primeiro módulo do sistema (No caso quando foi criada para *Predio\_A*). Ele não cria as entradas na *Type Library* quando se usa mais de uma interface, como nesse caso, onde foi criada *Espaco\_A*. Nesses casos, deve-se então adicionar manualmente uma interface (derivada de *ICpBusiness*) e uma co-class que implemente essa interface. Veja figura a seguir:



O nome da Interface deve iniciar com I. e a co-class deve possuir o mesmo nome porém sem o I na frente.

Para adicionara a interface ao *Implements* da co-class você deve selecionar a co-class, dar um clique de direita no painel a direita e escolher a opção *Insert Interface*, escolhendo então a interface *IEspacos\_A*, conforme tela abaixo:



#### Padrão de nomenclatura utilizado:

**Interface** = IXXX

**Co-Class** = XXX

**Classe Delphi** = TdmXXX

**Unit** = udmXXX

**Unit MetaQuery** = umqrXXX

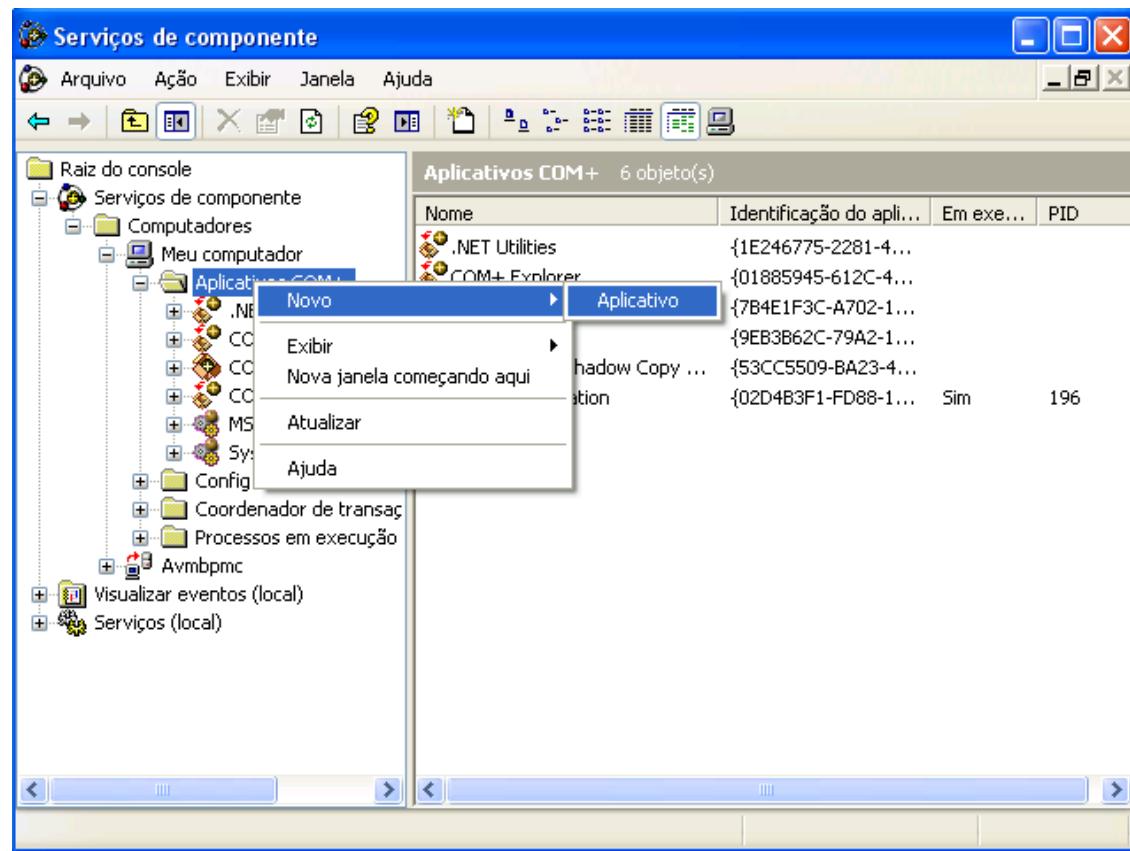
**Unit Cliente** = uCliXXX

O nome da *Interface* deve ser o mesmo que foi definido no gerador (veja o código gerado na unit):

Salve tudo e compile a aplicação, usando o menu Project -> Build

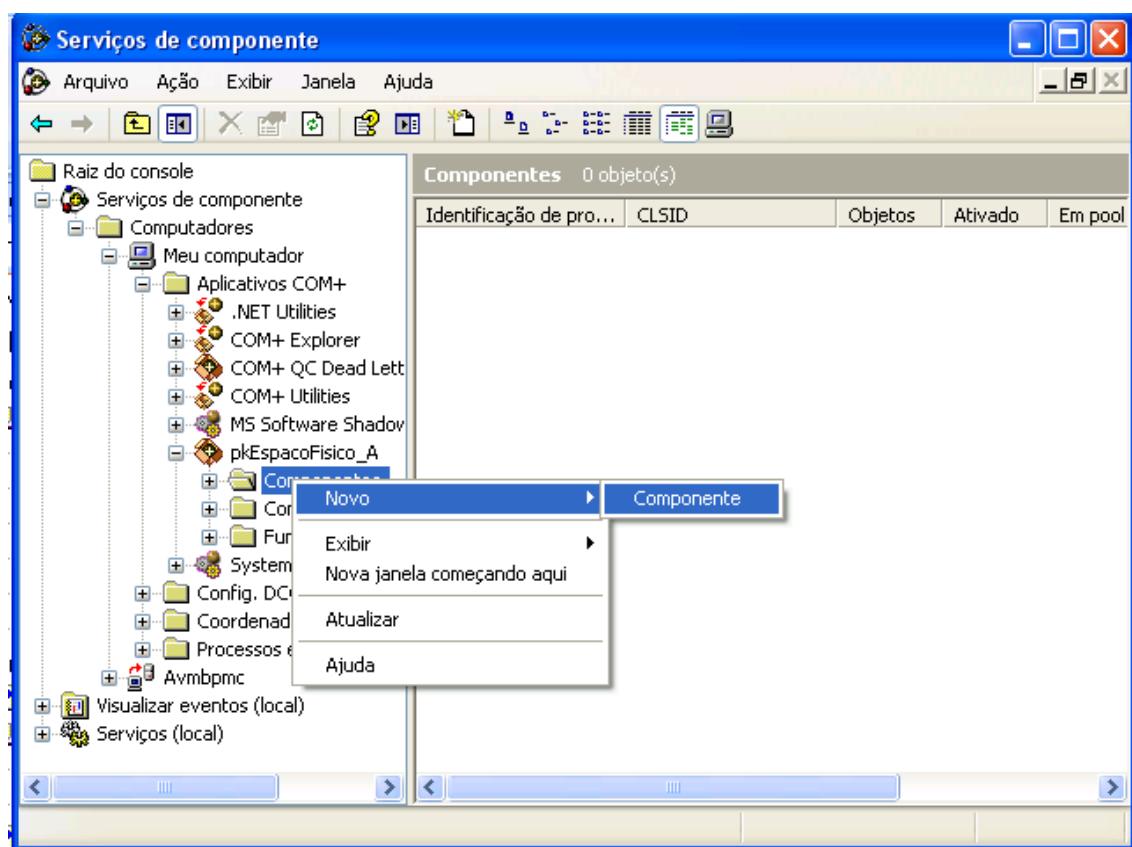
## 2.4 Instalando a aplicação servidora

Abra o menu *Serviços de Componente* (ou iniciar -> executar -> C:\WINDOWS\system32 \Com\comexp.msc), crie um novo aplicativo COM, clicando de direita em Aplicativos do COM+ (veja a figura).

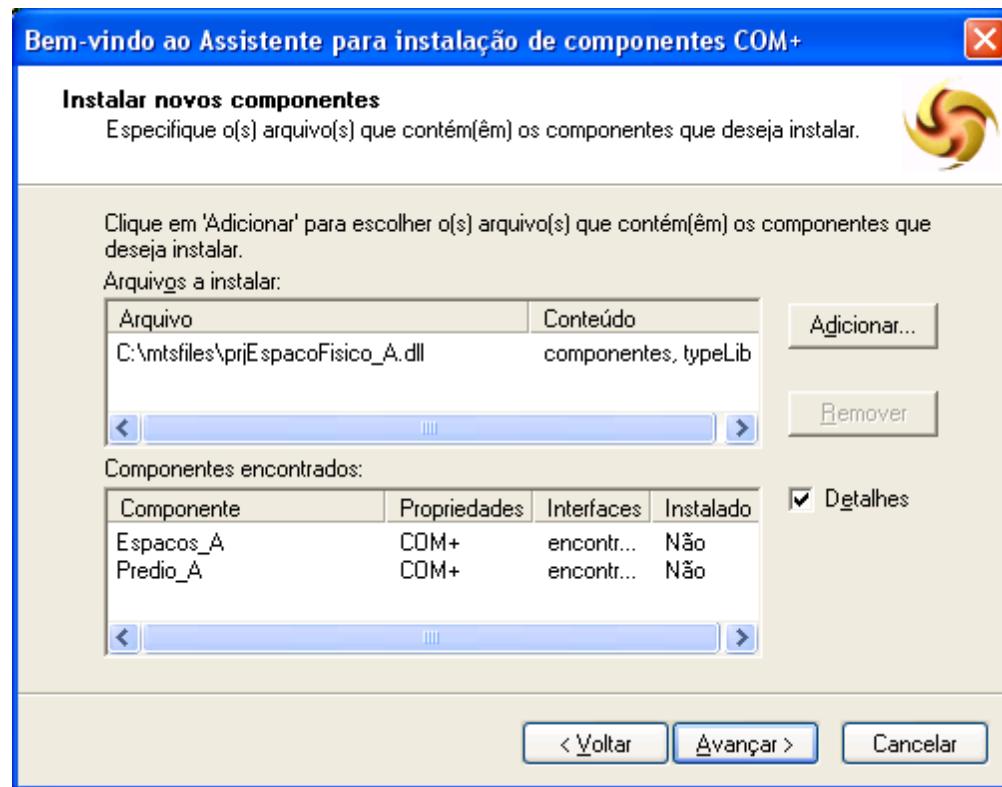


Clique em "Avançar", e depois "Criar um aplicativo vazio", e depois de o nome de "pkEspacoFisico\_A", clique em Avançar até chegar em Concluir.

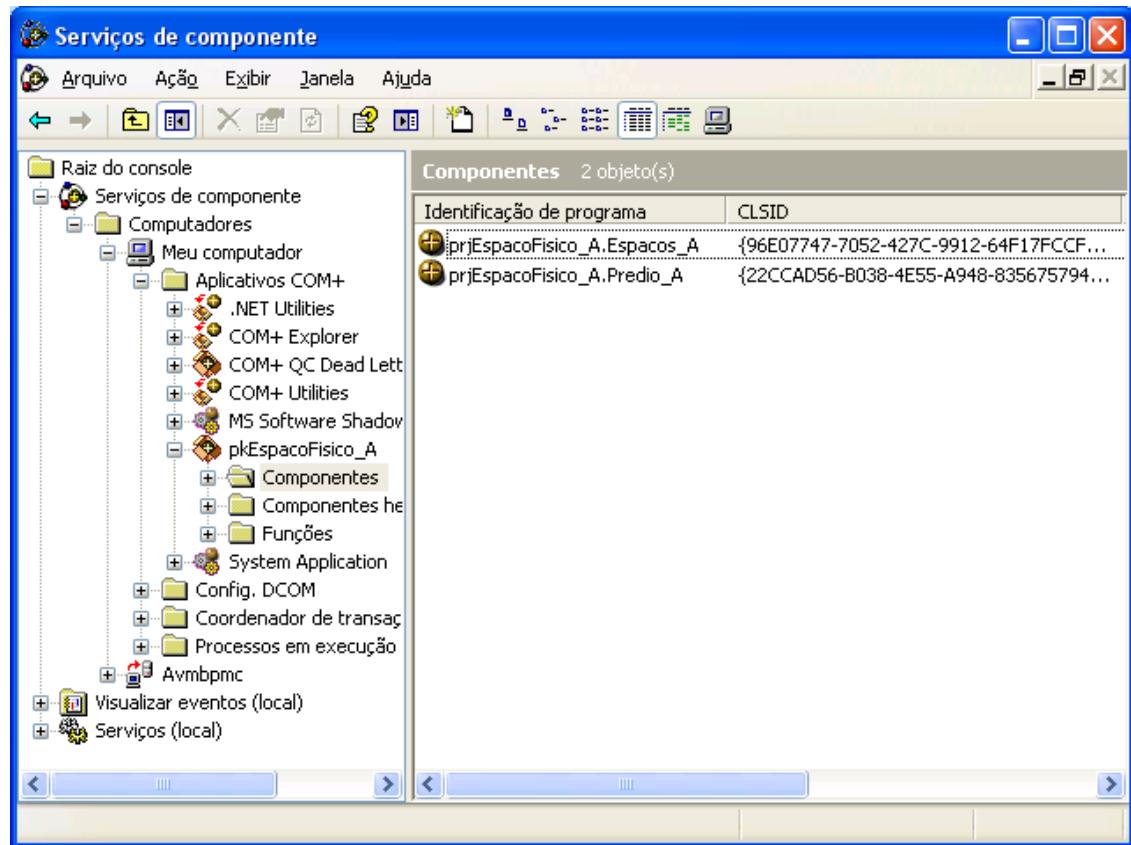
Abra o novo pacote na árvore, clique de direita em Componentes e escolha a opção "Novo" -> "Componente":



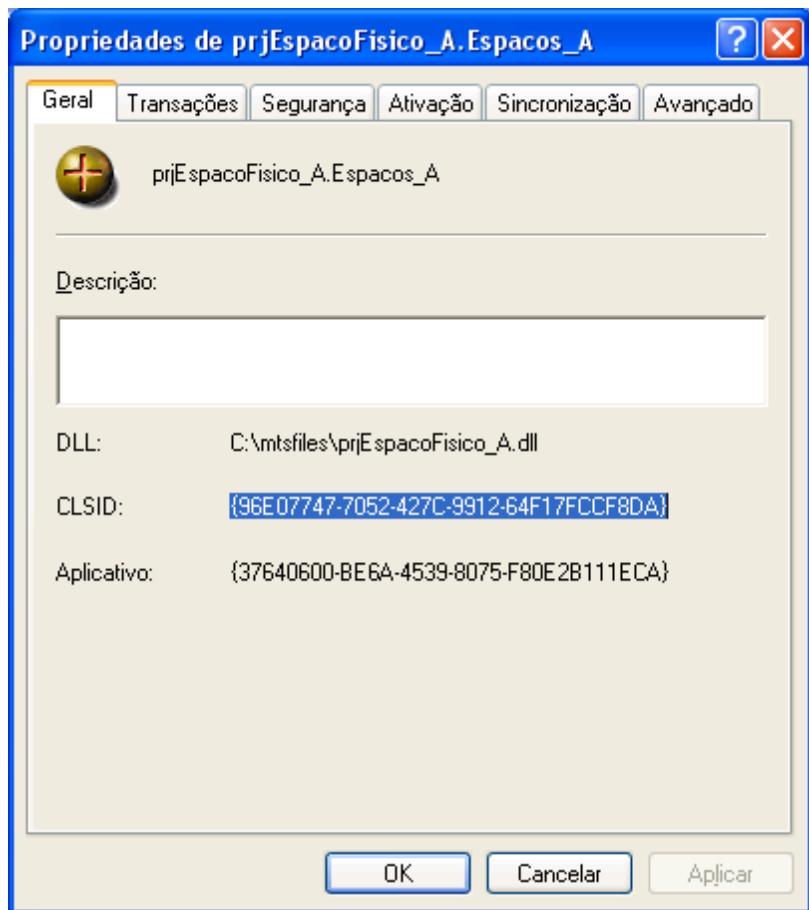
Clique em Avançar na primeira tela do assistente, depois escolha a opção Instalar novo Componente, na próxima tela clique em Adicionar e selecione então o arquivo recem compilado *prjEspacoFisico\_A.dll*.



Clique em *Avançar* e depois em *Concluir*. Observe que os componentes foram adicionados ao pacote:

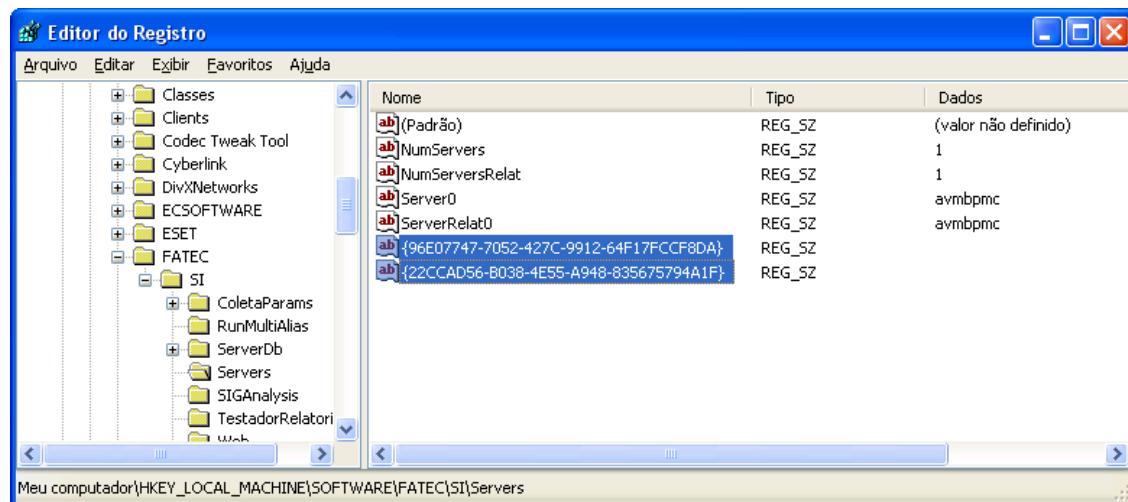


Dê um clique de direita no componente *prjEspacoFisico\_A.Espacos\_A* e escolha a opção Propriedades. Selecione o CLSID aperte Ctrl+C.



OBS: O CLSID é o GUID que identifica um componente COM. Esse ID será usado para rodar o projeto local.

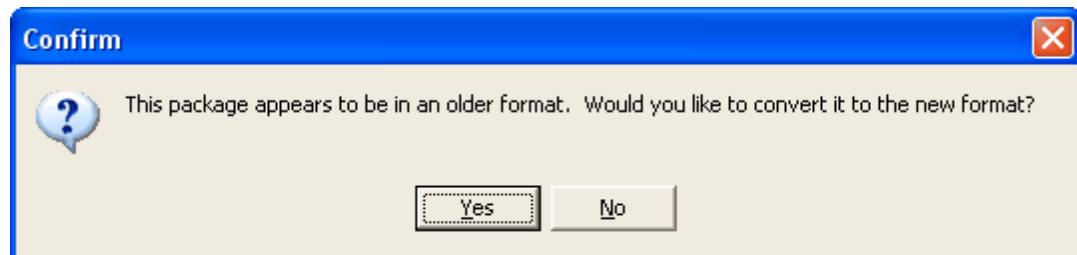
Abra o Editor do Registro do Windows (Iniciar - Executar - Regedit.exe), vá até a chave HKEY\_LOCAL\_MACHINE\Software\Fatec\SI\Servers. Esta chave indica qual o servidor de aplicações que as aplicações clientes deverão chamar. Para facilitar o desenvolvimento o SI\* possui um mecanismo que permite instanciar uma, ou mais, servidores local, caso contrário cada desenvolvedor teria que instalar todos os pacotes do sistema em sua máquina para poder depurar suas aplicações. Para rodar uma servidora local adicione um novo Valor de Seqüência e coloque no nome dele o CLSID que foi copiado nas propriedades da servidora. Faça o mesmo para prjEspacoFisico\_A.Predio\_A:



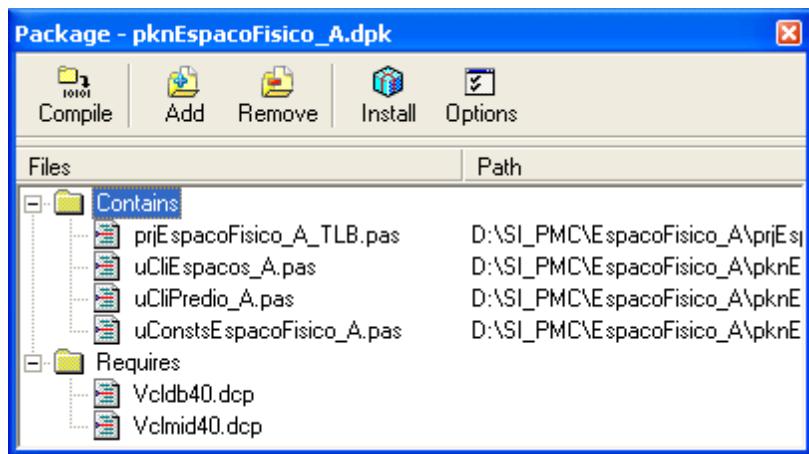
OBS: Essa configuração só é necessária para rodar o servidor COM+ localmente, sem que para isso seja necessário instalar o pacote no servidor de aplicação. Como é uma servidora nova, ela não existirá ainda no servidor de aplicações, portanto essa configuração é necessária para posteriormente testarmos as aplicações clientes. Além disso, essa configuração também é útil para depurar um servidor COM+.

## 2.5 Instalando o pkn

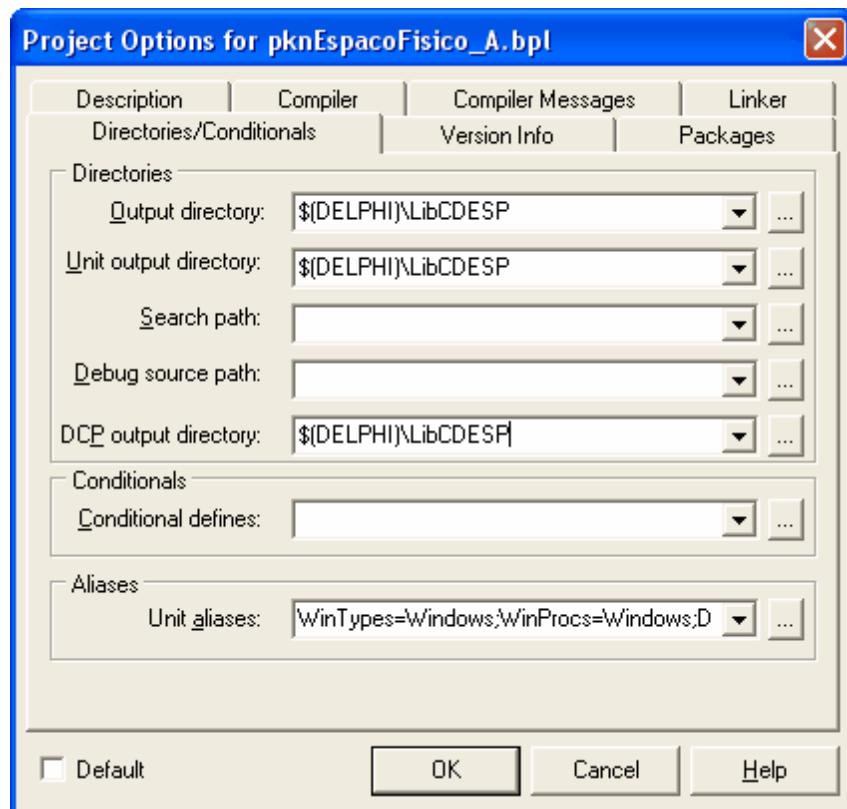
No Delphi, abra o arquivo “pknEspacoFisico\_A.dpk”, e confirme o update caso esteja usando Delphi 7:



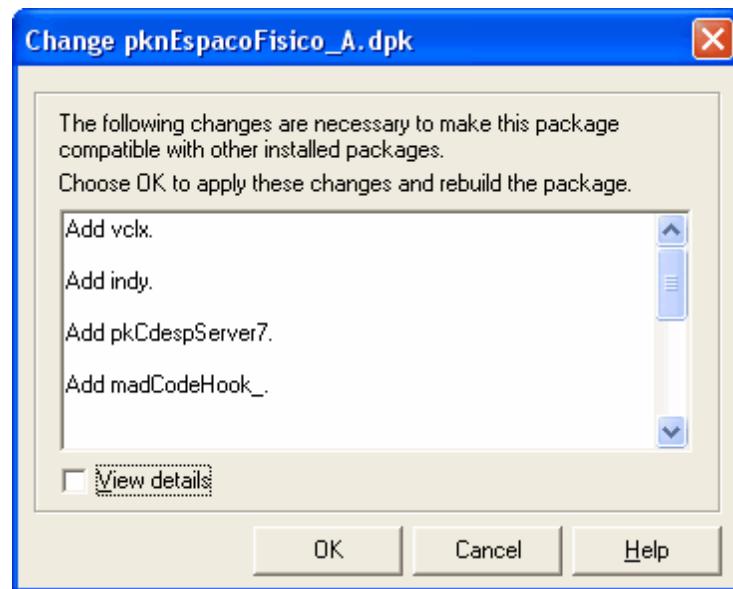
Limpe a seção "Requires", deixando somente os pacotes Vcl\*:



Clique em *Options* e preencha as opções como mostrado a seguir:



Clique em *Install* e confirme a inclusão dos novos pacotes no *Requires*.



Os novos componentes devem aparecer na paleta criada, prontos para o uso em aplicações.

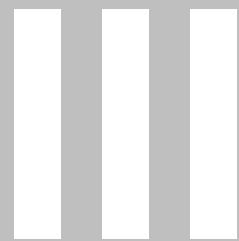


# **Top Level Intro**

This page is printed before a new  
top-level chapter starts

# **Part**

---



## 3 Criando uma aplicação cliente

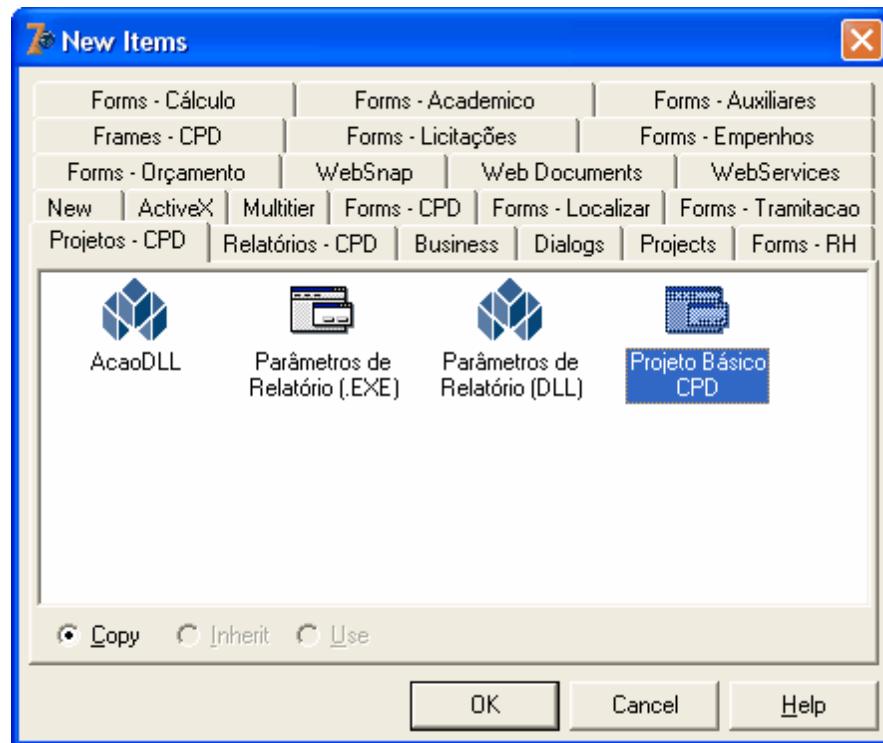
### 3.1 Criando uma nova aplicação (Cadastro de Prédios)

Faremos agora uma nova aplicação passo a passo para um servidor que já está pronto.

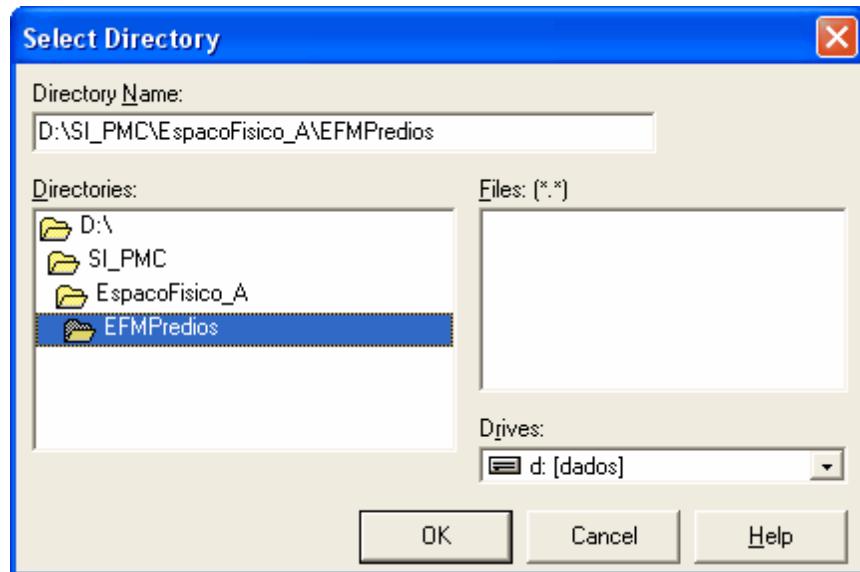
Crie um novo diretório seguindo a estrutura:

X:\XXXX\EspacoFisico\_A\EFMPredios

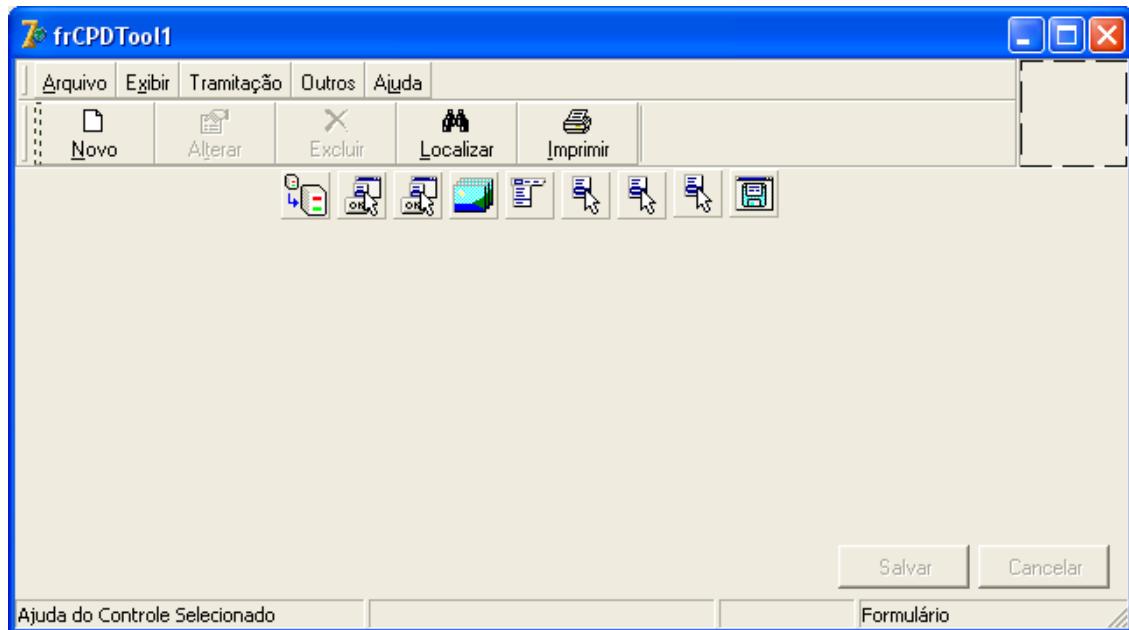
No Delphi, clique em File -> New -> Other > Projetos – CPD > Projeto Básico CPD.



Na janela que aparece selecione o diretório criado anteriormente.



Você verá na tela um formulário padrão, herdado de *TfrCPDTool* (do repositório), incluído no projeto.



Clique em "File -> Save Project As" e salve o projeto como "PredioEspacos". Selecione o formulário *frCPDTool1* (não *frCPDTool*), altere seu nome (propriedade Name) para "frPredios", clique em "File -> Save As" e salve-o como "ufrPredios". Selecione *dmClientCPDI*, altere seu nome para "dmPrediosEspacos" e salve-o como "udmPrediosEspacos".

Clique em "Project View -> Source". Localize a linha "if UsuarioAutorizado ....." e coloque // na frente dela, para que o Delphi ignore-a, ou seja, para que vire um "comentário":

```
//      if      UsuarioAutorizado(StrToIntDef(ParamStr(1),0),StrToIntDef(ParamStr(2),0)) then
```

OBS: Sempre que for testar alguma aplicação cliente por dentro do Delphi essa linha deve ser comentada. Após as alterações das aplicações, retire as // da linha. Esse comando evita que as aplicações do SI\* sejam executadas por fora do GCANavegacao.exe.

Abra o formulário principal, aperte Alt+F11 ou "File - Use Unit" e selecione a unit do DataModule (udmPrediosEspacos).

No evento OnCreate do formulário **principal** digite as seguintes informações:

```
procedure TfrPredios.FormCreate(Sender: TObject);
begin
  inherited;
  // CliPrincipal := dmXXX.CliXXX;
  FNomeExecutavel := 'EFMPredios.exe';
  AddDataModule(dmPrediosEspaco);
end;
```

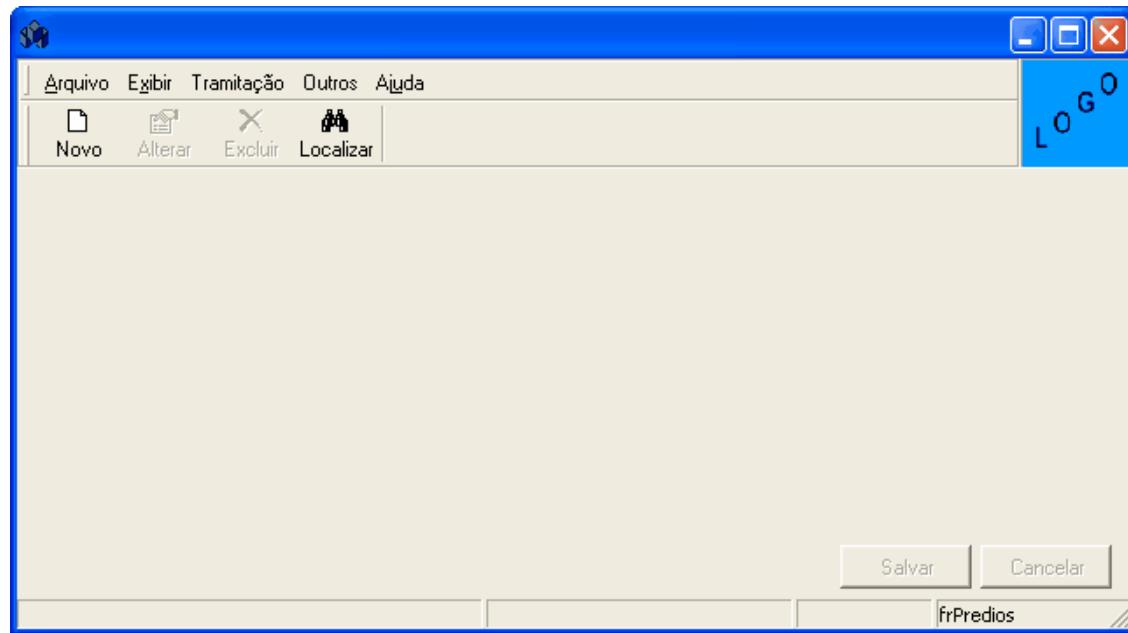
OBS: Ainda não colocaremos a informação do CliPrincipal, por enquanto deixe comentada a linha.

## 3.2 Adicionando componentes ao DataModule

Abra o *dmPrediosEspaco* e coloque nele os componentes *CliBusiness* mostrados na figura a seguir.



Você já pode executar a aplicação para testá-la, através do menu "CDESP Experts -> Multitier Tools - Run Multi Alias".



### 3.3 Paleta Bs Standard



A paleta Bs Standard contém os componentes básicos para criação de formulários de dados no SI\*. A maioria dos componentes se conecta a um componente *CliBusiness*.

Coloque no formulário um componente *BsLabelTop* (paleta Bs Additional). Coloque um *BsPageControl* no formulário. Adicione duas abas neste PageControl, a primeira coloque o nome como “*tbshPredio*”. No evento *onShow* do formulário adicione o seguinte código:

```
procedure TfrPredios.FormShow(Sender: TObject);
begin
  inherited;
  tbshPredio.Caption := dmPrediosEspaco.CliPredio_A1.DisplayLabelSingular;
end;
```

Na aba Prédios coloque novos componentes *BsContainer* e configure suas propriedades como mostrado a seguir:

```
object BsNumPredio: TBsContainer
  BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
  BsClientData.FieldName = 'NUM_PREDIO'
  BsControlType = bstEdit
  object BsNumPredioEdit: TBsEdit
    BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
    BsClientData.FieldName = 'NUM_PREDIO'
    BsClientData.BsDataSource = 'dsGetRecord'
  end
end
object BsCBloco: TBsContainer
  BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
```

```

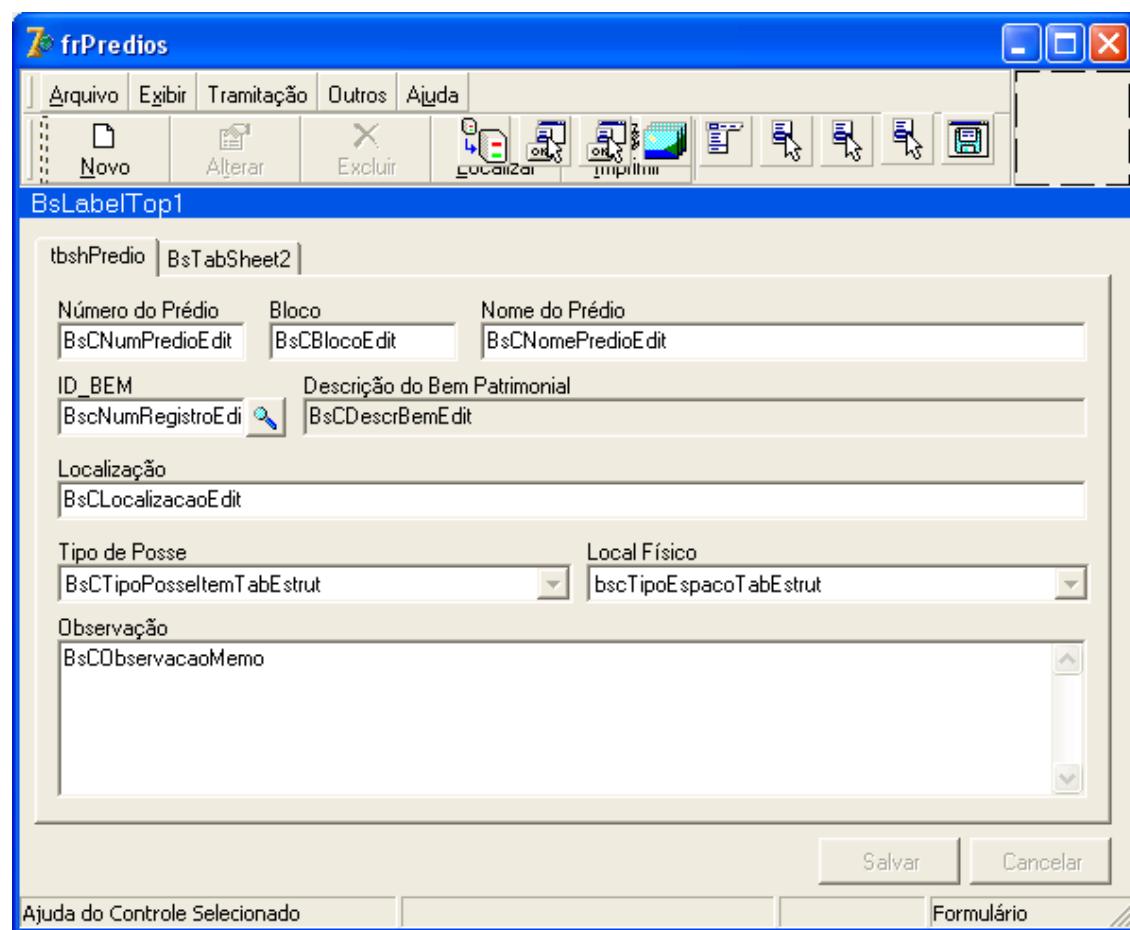
        BsClientData.FieldName = 'BLOCO'
        BsControlType = bstEdit
        object BsCBlocoEdit: TBsEdit
            BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
            BsClientData.FieldName = 'BLOCO'
            BsClientData.BsDataSource = 'dsGetRecord'
        end
    end
    object BsCNomePredio: TBsContainer
        BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
        BsClientData.FieldName = 'NOME_PREDIO'
        BsControlType = bstEdit
        object BsCNomePredioEdit: TBsEdit
            BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
            BsClientData.FieldName = 'NOME_PREDIO'
            BsClientData.BsDataSource = 'dsGetRecord'
        end
    end
    object BscNumRegistro: TBsContainer
        BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
        BsClientData.FieldName = 'ID_BEM'
        BsControlType = bstEdit
        Localizar = True
        object BscNumRegistroEdit: TBsEdit
            BsClientData.BsClient = dmPrediosEspaco.CliBemPatrimonial1
            BsClientData.FieldName = 'NUM_REGISTRO'
            BsClientData.BsDataSource = 'dsGetRecord'
        end
    end
    object BsCDescrBem: TBsContainer
        BsClientData.BsClient = dmPrediosEspaco.CliBemPatrimonial1
        BsClientData.FieldName = 'DESCR_BEM'
        BsControlType = bstEdit
        object BsCDescrBemEdit: TBsEdit
            ParentColor = True
            ReadOnly = True
            BsClientData.BsClient = dmPrediosEspaco.CliBemPatrimonial1
            BsClientData.FieldName = 'DESCR_BEM'
            BsClientData.BsDataSource = 'dsGetRecord'
        end
    end
    object BsCLocalizacao: TBsContainer
        BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
        BsClientData.FieldName = 'LOCALIZACAO'
        BsControlType = bstEdit
        object BsCLocalizacaoEdit: TBsEdit
            BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
            BsClientData.FieldName = 'LOCALIZACAO'
            BsClientData.BsDataSource = 'dsGetRecord'
        end
    end
    object BsCTipoPosseItem: TBsContainer
        BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
        BsClientData.FieldName = 'TIPO_POSSE_ITEM'
        BsControlType = bstTabEstrut
        object BsCTipoPosseItemTabEstrut: TBsTabEstrut
            BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
            BsClientData.FieldName = 'TIPO_POSSE_ITEM'
        end
    end
    object bscTipoEspaco: TBsContainer

```

```

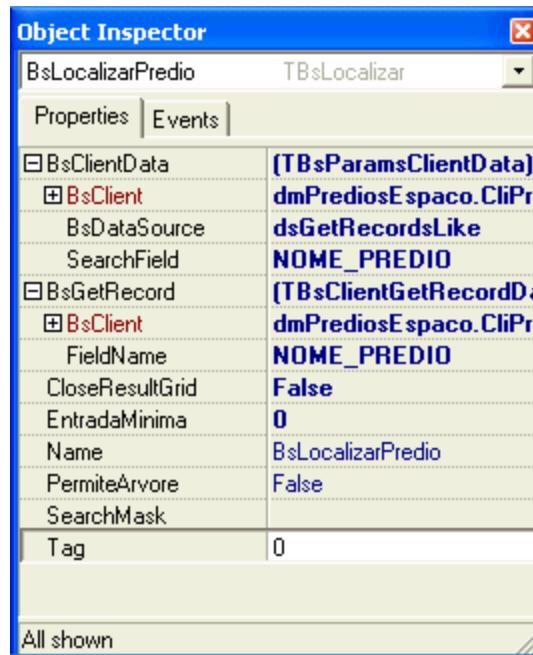
        BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
        BsClientData.FieldName = 'LOCAL_FISICO_ITEM'
        BsControlType = bstTabEstrut
        object bscTipoEspacoTabEstrut: TBsTabEstrut
            BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
            BsClientData.FieldName = 'LOCAL_FISICO_ITEM'
        end
    end
    object BsCObservacao: TBsContainer
        BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
        BsClientData.FieldName = 'OBSERVACAO'
        BsControlType = bstMemo
        object BsCObservacaoMemo: TBsMemo
            BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
            BsClientData.FieldName = 'OBSERVACAO'
            BsClientData.BsDataSource = 'dsGetRecord'
        end
    end

```



### 3.4 BsLocalizar

Coloque no formulário um componente *BsLocalizar* da paleta *BsLocalizar*, chamando-o de “*BsLocalizarPredio*”. Configure suas propriedades como mostrado a seguir:



Dê um duplo clique no botão Localizar e digite o seguinte código:

```
procedure TfrPredios.LocalizarExecute(Sender: TObject);
begin
  if BsLocalizarPredio.Execute then
    inherited;
end;
```

Implemente o código do método *PosicionaRegistro* da seguinte forma:

```
procedure TfrPredios.PosicionaRegistro;
begin
  with dmPrediosEspaco do
  begin
    CliBemPatrimonial1.GetRecordPK(CliPredio_A1.FieldName('ID_BEM').AsInteger,
      BsLabelTop1.Text := CliPredio_A1.FieldName('NOME_PREDIO').DisplayLabel +
        CliPredio_A1.FieldName('NOME_PREDIO').AsString;
  end;
  inherited;
end;
```

### 3.5 Habilitando / Desabilitando controles

No método *EnableDisableControls* coloque o seguinte código:

```
procedure TfrPredios.EnableDisableControls(aEnable: Boolean);
begin
  inherited;
  BsCNumPredio.Enabled := aEnable;
  BsCBloco.Enabled := aEnable;
  BsCNomePredio.Enabled := aEnable;
  BscNumRegistro.Enabled := aEnable;
  BscDescrBem.Enabled := aEnable;
  BsCLocalizacao.Enabled := aEnable;
  BsCTipoPosseItem.Enabled := aEnable;
  bscTipoEspaco.Enabled := aEnable;
  BsCObservacao.Enabled := aEnable;
end;
```

Note que este método é declarado como virtual na classe pai (TfrCPDTool), portanto não há a necessidade de chamá-lo na aplicação, somente implementá-lo.

Faça o teste: abra a aplicação e note que os controles estão desabilitados, clique em novo, cancelar, editar, cancelar.

Dica: Como só existem *TBsContainers* na tela, o código implementado em *EnableDisableControls* pode ser escrito da seguinte forma:

```
procedure TfrPredios.EnableDisableControls(aEnable: Boolean);
var i: integer;
begin
  inherited;
  for i:=0 to ComponentCount-1 do
    if Components[i] is TBsContainer then
      (Components[i] as TBsContainer).Enabled := aEnable;
end;
```

### 3.6 Manipulação de registros

Iremos implementar os códigos para Inserir, Editar e Excluir registros.

Dê um duplo clique no botão Novo e coloque o seguinte código:

```
procedure TfrPredios.NovoExecute(Sender: TObject);
begin
  inherited;
  dmPrediosEspaco.CliPredio_A1.Append(True);
end;
```

Dê um duplo clique no botão Alterar e coloque o seguinte código:

```
procedure TfrPredios.AlterarExecute(Sender: TObject);
begin
  inherited;
  dmPrediosEspaco.CliPredio_A1.Edit;
end;
```

Adicione a unit *UtilCPD* no *uses*. Dê um duplo clique no botão Excluir e coloque o seguinte código:

```
procedure TfrPredios.ExcluirExecute(Sender: TObject);
const CONFIRMA_EXCLUSAO = 2;
begin
  with dmPrediosEspaco do
    if(MessageDlg(CONFIRMA_EXCLUSAO, mtcConfirmacao, [mbcSim,mbcNao], 0, '' ,
      [CliPredio_A1.FieldName('NOME_PREDIO').Value]) = mrYes) then
    begin
      AtualizacaoEfetuada := CliPredio_A1.Delete;
      if AtualizacaoEfetuada then
        inherited;
    end;
end;
```

### 3.7 Salvando e Cancelando

Dê um duplo clique no botão Salvar e digite o seguinte código:

```
procedure TfrPredios.SalvarExecute(Sender: TObject);
begin
  with dmPrediosEspaco do
    begin
      case frmState of
        fstInsert : AtualizacaoEfetuada := CliPredio_A1.Insert;
        fstEdit : AtualizacaoEfetuada := CliPredio_A1.Update;
      end;
      if AtualizacaoEfetuada then
        inherited;
    end;
end;
```

Dê um duplo clique no botão Cancelar e digite o seguinte código:

```
procedure TfrPredios.CancelarExecute(Sender: TObject);
begin
  dmPrediosEspaco.CliPredio_A1.Cancel;
  inherited;
end;
```

Implemente o código do método *CloseDataSets* da seguinte forma:

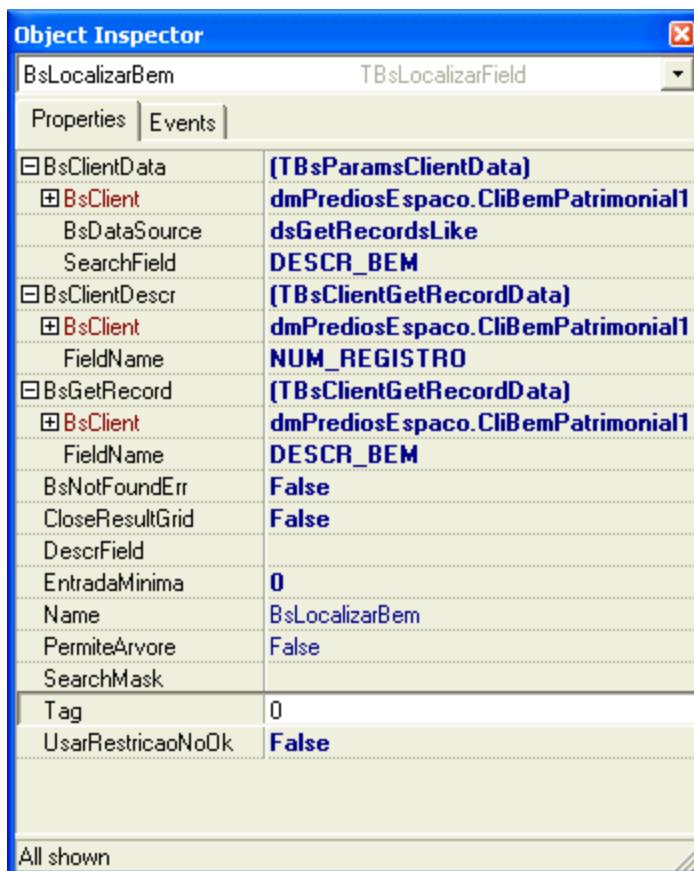
```

procedure TfrPredios.CloseDataSets;
begin
  inherited;
  With dmPrediosEspaco do
  begin
    CliEspacos_A1.CliDsGetRecord.Close;
    CliBemPatrimonial1.CliDsGetRecord.Close;
  end;
end;

```

### 3.8 Localizar Bem Patrimonial

Coloque no formulário um componente *BsLocalizarField* da paleta *BsLocalizar*, chamando-o de “BsLocalizarBem”. Configure suas propriedades como mostrado a seguir:



Dê um duplo no botão localizar "Bem Patrimonial" e digite o seguinte código:

```

procedure TfrPredios.BscNumRegistrobtnLocalizarClick(Sender: TObject);
begin
  inherited;
  if BsLocalizarBem.Execute then
    dmPrediosEspaco.CliPredio_A1.FieldByName('ID_BEM').Value :=
      dmPrediosEspaco.CliBemPatrimonial1.FieldByName('ID_BEM').Value;
end;

```

A aplicação já está pronta para inserir dados na tabela PREDIOS\_A. Faça o teste. Clique em novo, preencha os dados, salve. Localize, edite, salve ...

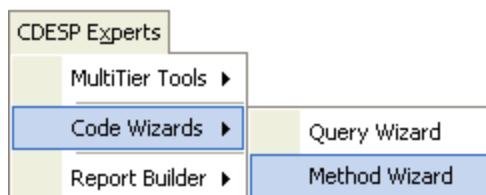
### 3.9 Criando métodos customizados

Até aqui utilizamos somente os métodos pré-definidos por *ICpBusiness* (*Insert*, *Update*, *GetRecordPK* ..), interface base das duas interfaces que criamos. Muitas vezes precisamos criar novos métodos que executam determinada tarefa no servidor ou no banco de dados. Esses métodos podem ser facilmente criados utilizando o *Method Wizard* no Delphi. *Method Wizard* é um assistente desenvolvido que permite criar rapidamente novos métodos, adicionando inclusive o código necessário no *TClBusiness*.

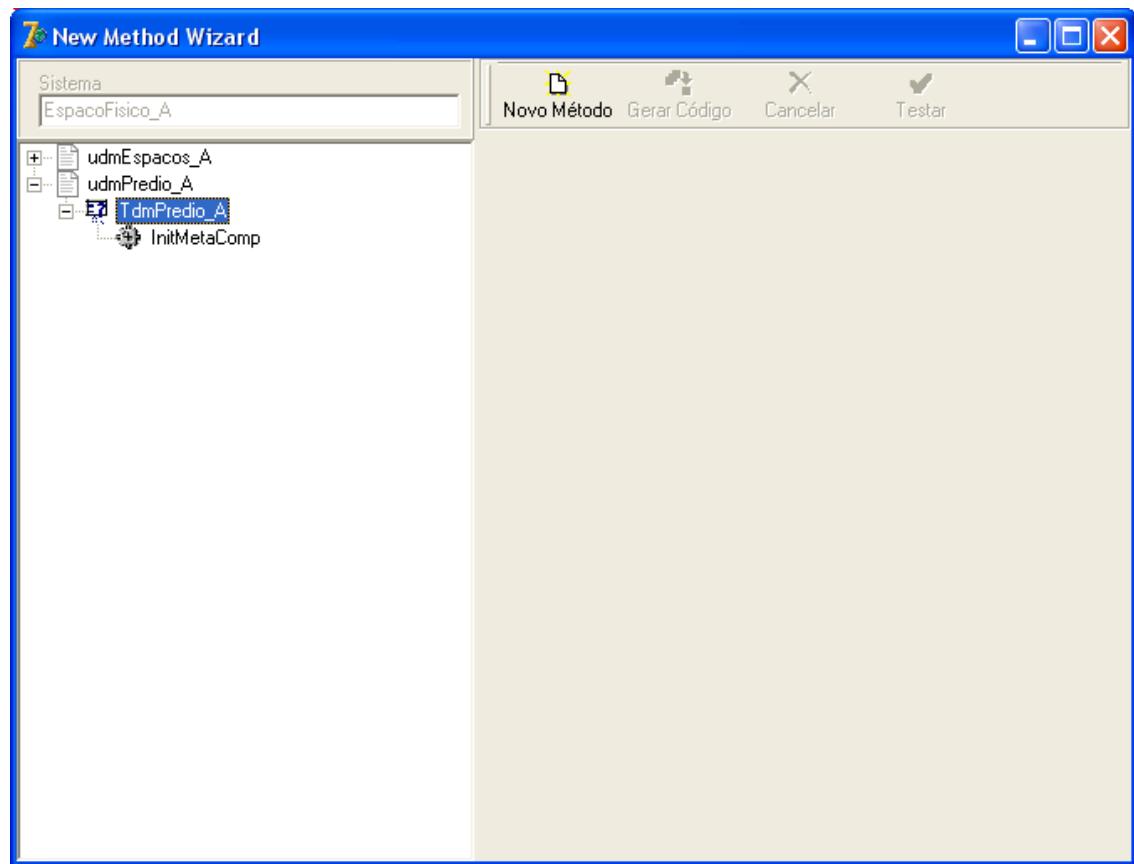
Na nossa aplicação, iremos criar um atalho para a criação de "Espaços" dentro do prédio que foi cadastrado, porém, numa segunda aba, deseja-se mostrar todos os espaços que estão vinculados ao prédio. Ou seja, localiza-se o prédio, clica-se na segunda aba e serão listados todos os espaços cadastrados.

Para isso, teremos que criar um método customizado na servidora, que retorne esses dados.

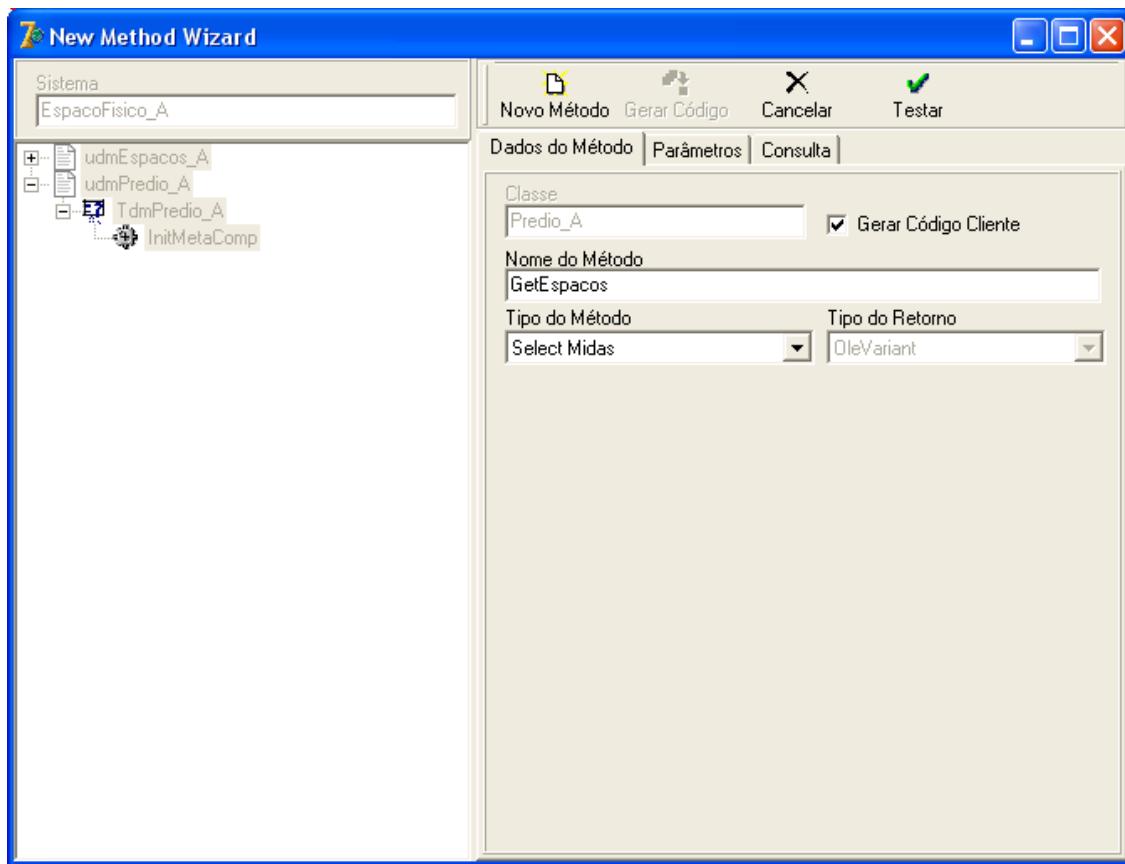
No Delphi, abra a servidora "prjEspacoFisico\_A.dpr" e clique no menu CDESP Experts -> Code Wizards -> Method Wizard:



Selecione a classe TdmPredio\_A e clique no botão Novo Método.



Dê o nome de *GetEspacos* para o método e defina seu tipo como *Select Midas*:



a

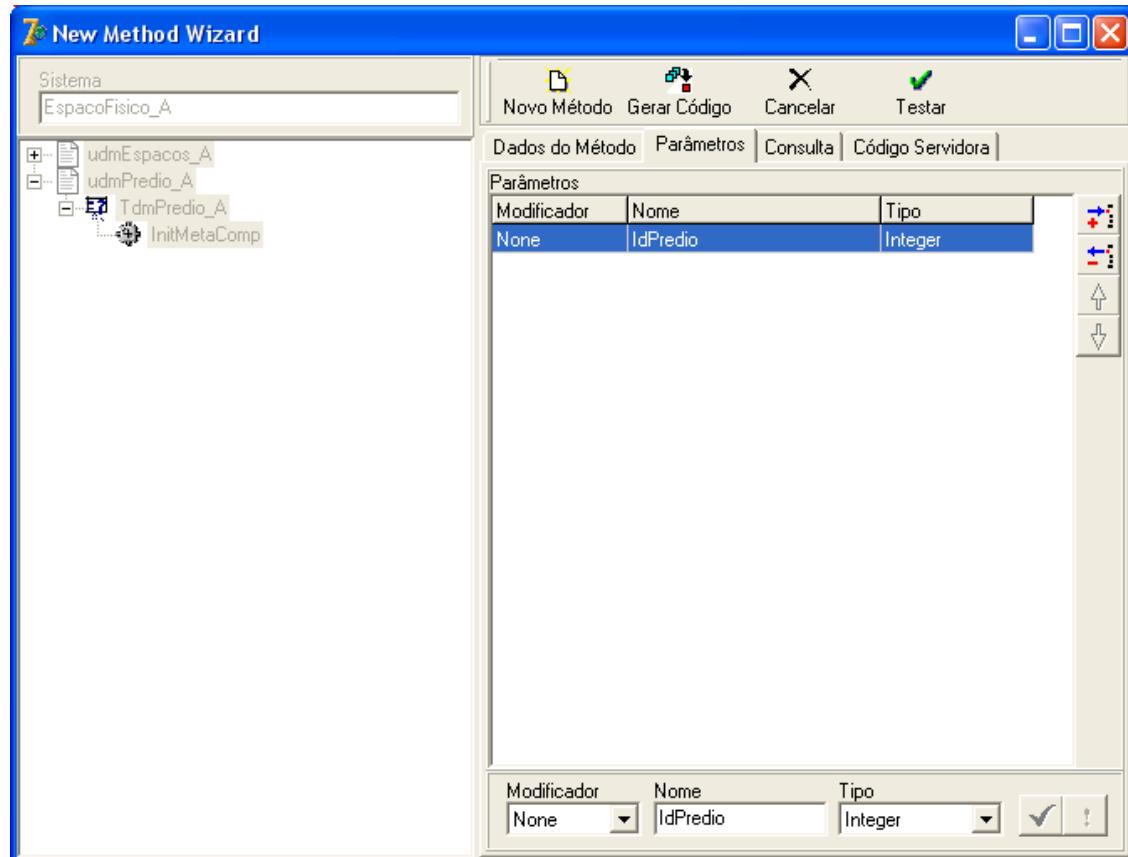
### Tipo do método:

- **Select** : Retorna para a função cliente o tipo especificado em *Tipo do Retorno*;
- **Select Midas** : Idêntico ao select, sendo que o tipo de retorno é um *OleVariant* mapeado para um *TClientDataSet* no componente *CliBusiness*;
- **Select Localizar** : Idêntico a um select *Midas*, sendo que o método gerado seguirá o padrão de um método de busca para componentes “*Localizar*”, com parâmetros pré-definidos. (Ver seção [BsLocalizar](#)<sup>[37]</sup>)
- **Update/Insert** : Métodos que utilizaram comandos *Update/Insert*. Método mapeado do componente *CliBusiness* retorna *True* se comando foi executado com sucesso;

Deixe marcada a opção "Gerar Código Cliente", para o *Wizard* gerar o código necessário no *TCliPredio\_A*, para interpretar os dados retornados. Ou seja, criar um *TClientDataSet* para receber os dados e um *TDataSource* para ser vinculado aos componentes visuais da aplicação. Caso deseja-se criar algum método que só será usado internamente dentro da servidora, não é necessário "Gerar Código Cliente".

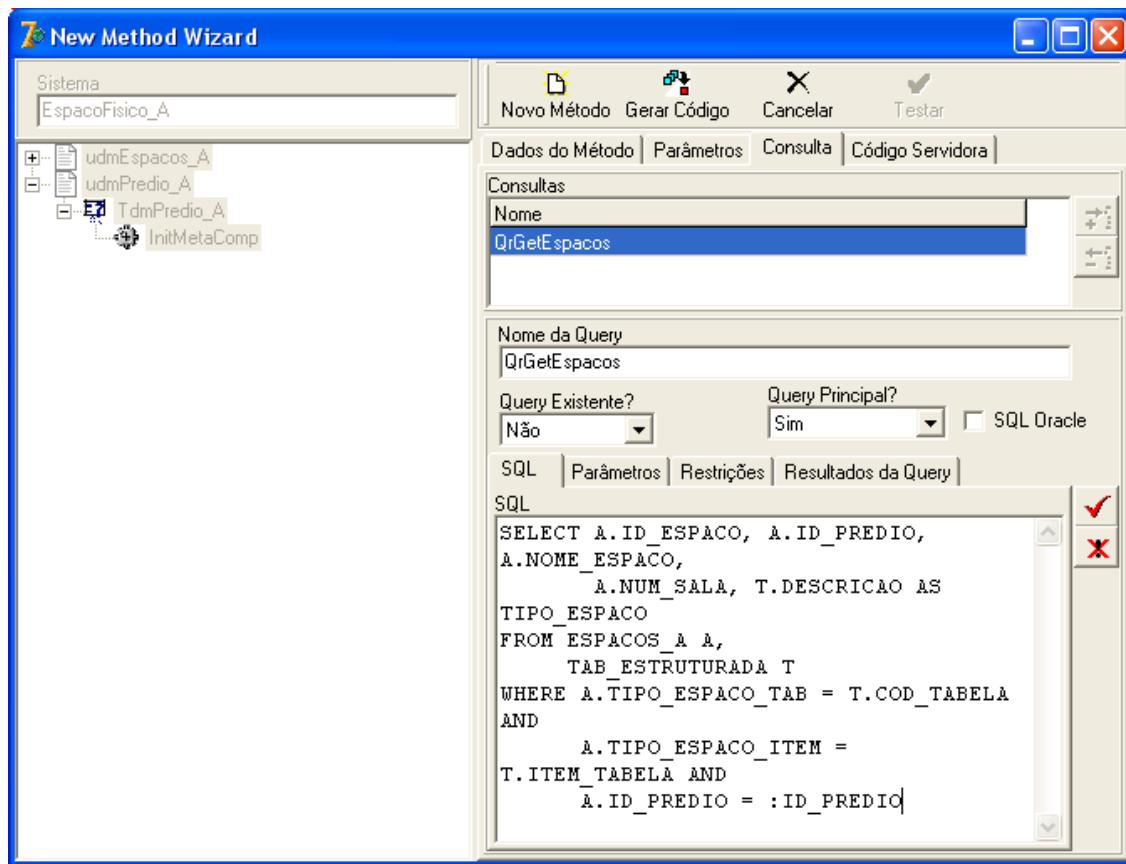
Na aba parâmetros, vamos definir qual vai ser o parâmetro da consulta SQL, no caso

IdPredio do tipo *Integer*:

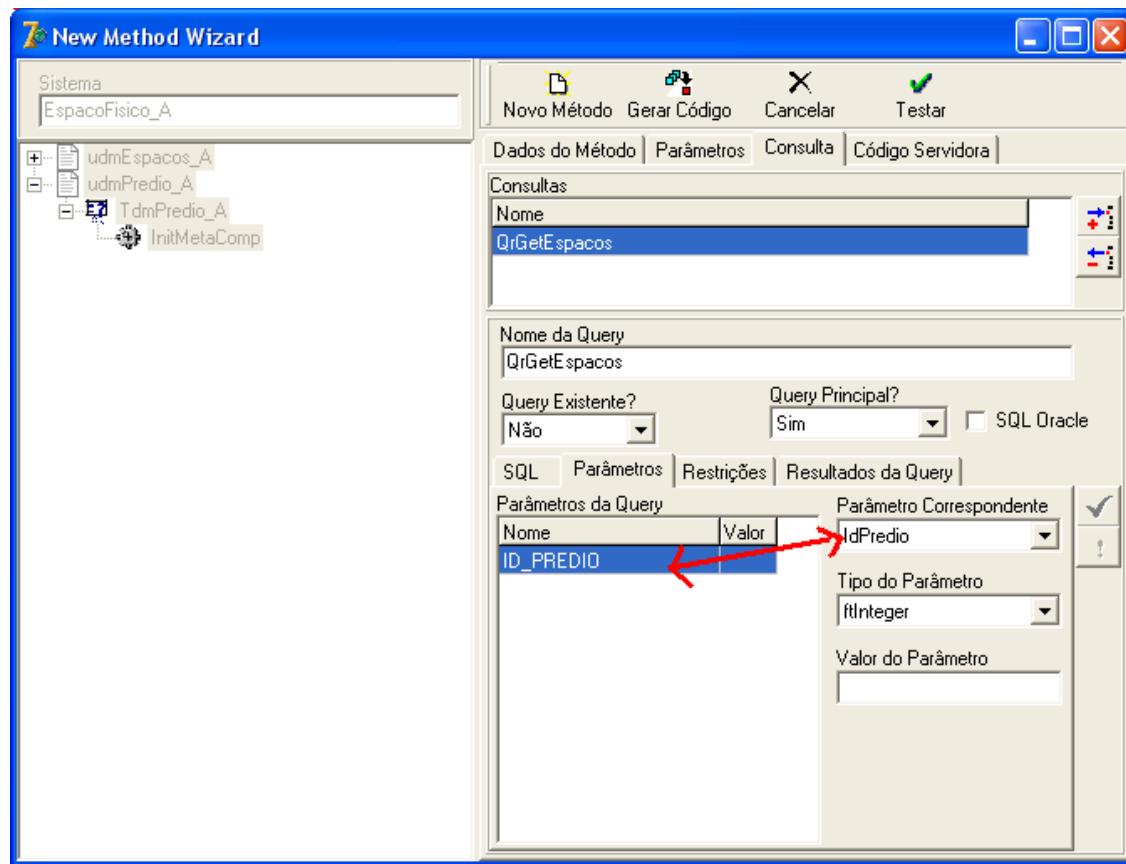


Na aba Consulta, utilize a seguinte consulta SQL:

```
SELECT A.ID_ESPACO, A.ID_PREDIO, A.NOME_ESPACO,
       A.NUM_SALA, T.DESCRICAO AS TIPO_ESPACO
  FROM ESPACOS_A A,
       TAB_ESTRUTURADA T
 WHERE A.TIPO_ESPACO_TAB = T.COD_TABELA AND
       A.TIPO_ESPACO_ITEM = T.ITEM_TABELA AND
       A.ID_PREDIO = :ID_PREDIO
```



Após digitar a consulta, clique em para confirmar. Na guia parâmetros inferior, certifique que os parâmetros estão mapeados corretamente, ou seja, parâmetro do método com parâmetro da consulta SQL:



Para finalizar, clique no botão Gerar Código, e após a confirmação da geração, feche essa janela. Isso criará uma nova classe *TMetaQuery* (*TQrGetEspacos*, na unit *umqrPredio\_A*), que guardará a instrução SQL (*TMetaQuery*) é instanciada na implementação do método *GetEspacos* e repassada ao servidor contexto. O retorno do contexto é repassado diretamente para o cliente como resultado da execução do método.

*TMetaQuery* é uma classe que permite definir uma instrução SQL independente de mecanismo de acesso a dados, para que possa ser facilmente passada de um aplicação servidora para o contexto.

Veja a seguir a implementação do método gerado e a definição da *TMetaQuery*:

```

function TdmPredio_A.GetEspacos( IdPredio: Integer; CtxSGCA: OleVariant;
  const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant;
var aCtx: IBsContextoDb;
  MetaQr: TQrGetEspacos;
begin
try try
  aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
  MetaQr := TQrGetEspacos.Create;
  try
    MetaQr.ParamByName('ID_PREDIO').Value := IdPredio;
    result := aCtx.OpenBsQuery( MetaQr.PackForCom(CtxSGCA), CtxSGCA, Errors);
  finally
    MetaQr.Free;
  end;
except
  on E: Exception do
    GetCpErrorsParam(E, Errors, 176, Name, ['GetEspacos'], CtxSGCA);
end;
finally
  SetComplete;
end;
end;

```

*CtxDb* é uma instância da interface *IBsContextoDb* (ver seção [Contexto](#)<sup>9</sup> e [Controle de Transações](#)<sup>23</sup>) que representa a conexão com o servidor de contexto, quando o cliente chama um método servidor é passado **nil** como parâmetro, a função *ConsisteBsContextoDb* verifica este parâmetro é uma conexão válida e, caso negativo, obtém uma nova instância do contexto. Assim outros métodos que sejam chamados a partir deste podem utilizar essa mesma conexão.

Depois a *TMetaQuery* é instanciada e os parâmetros são definidos. Essa *TMetaQuery* é empacotada (utilizando a função pública *PackForCom(CtxSGCA: OleVariant)*) e passada para o método *OpenBsQuery* do contexto, este método executa a query no banco e retorna um pacote *Midas*, o qual pode ser atribuído para o propriedade Data de objetos da classe *TClientDataset*.

Caso ocorra algum erro (exceção) o método *GetCpErrorsParam* empacota os dados do erro na variável *Errors* (ver seção [manipulação de erros](#)<sup>25</sup>), do tipo *ICpErrors*. “176” é o identificador da mensagem de erro a ser exibida (o valor 176 identifica a mensagem “*Não foi possível executar o método %s*”). *Name* é o objeto servidor que gerou o erro, e *GetEspacos* é o nome do método.

```

constructor TQrGetEspacos.Create;
begin
  inherited;
  ServerName := 'dmPredio_A';
  QueryName := 'QrGetEspacos';
  SQL.Text :=
    'SELECT A.ID_ESPACO, A.ID_PREDIO, A.NOME_ESPACO, ' + #13#10 +
    '      A.NUM_SALA, T.DESCRICAO AS TIPO_ESPACO' + #13#10 +
    'FROM ESPACOS_A A,' + #13#10 +
    '      TAB_ESTRUTURADA T' + #13#10 +
    'WHERE A.TIPO_ESPACO_TAB = T.COD_TABELA AND' + #13#10 +
    '      A.TIPO_ESPACO_ITEM = T.ITEM_TABELA AND' + #13#10 +
    '      A.ID_PREDIO = :ID_PREDIO';
  ParamByName('ID_PREDIO').DataType := ftInteger;
end;

```

O construtor da classe *TQrGetEspacos* informa o SQL a ser executado, o nome da servidora, nome da query e os tipos dos parâmetros. Estes dados ficam predefinidos para facilitar o reuso de queries em diferentes métodos servidores.

No componente cliente (TCliPredios\_A) é criado o método *GetEspacos* com apenas o parâmetro necessário a query (IdPredio). Os demais parâmetros são abstraídos. Além, o retorno do método já é repassado a um TClientDataSet. Por padrão o nome dos TClientDataSet serão cds + nome do método, no exemplo, cdsGetEspacos.

```

function TCliPredio_A.GetEspacos( IdPredio: Integer): Boolean;
var Errors: ICpErrors;
  Data: OleVariant;
begin
  cdsGetEspacos.Close;
  Data := (RemoteBusiness as IPredio_ADisp).GetEspacos(IdPredio, GetCtxSGCA, ni);
  result := SemErro(Errors);
  if result then
    cdsGetEspacos.Data := Data;
    TrataCpErrors(Errors);
end;

```

Note também que no TCli também é gerado um *TClientDataSet* para o desempacotamento e manipulação dos dados e um *TDataSource* para vinculação a componentes *data-aware* como mostrado a seguir.

```

type
  TCliPredio_A = class(TCliBusiness)
private
  FCdsGetEspacos: TClientDataSet;
  FDsGetEspacos: TDataSource;
  { Private declarations }
protected
  procedure CreateGetEspacos;
  procedure CreateDataSets; override;
  { Protected declarations }
published
  function GetCdsGetEspacos: TClientDataSet;
  function GetDsGetEspacos: TDataSource;
public
  function GetEspacos( IdPredio: Integer): Boolean;
  property cdsGetEspacos: TClientDataSet read GetCdsGetEspacos;
  property dsGetEspacos: TDataSource read GetDsGetEspacos;
  procedure ActiveCliIDs(aCliIDs: TClientDataSet); override;
  constructor Create(aOwner: TComponent); override;
published
  { Published declarations }
end;

```

Todos esses códigos foram gerados automaticamente pelo *Method Wizard*.

### 3.9.1 Adicionado método na TLB

O *Method Wizard* gera todo o código necessário na cliente e servidora, porém não atualiza a *Type Library*. Você deve então abrir o editor de *Type Library* e adicionar o método *GetEspacos* com seus devidos parâmetros na interface IPredio\_A.

A forma mais simples de fazer essa tarefa é a seguinte:

Abra a udmPredio\_A e selecione a assinatura do método GetEspacos, e dê um Ctrl + C:

```

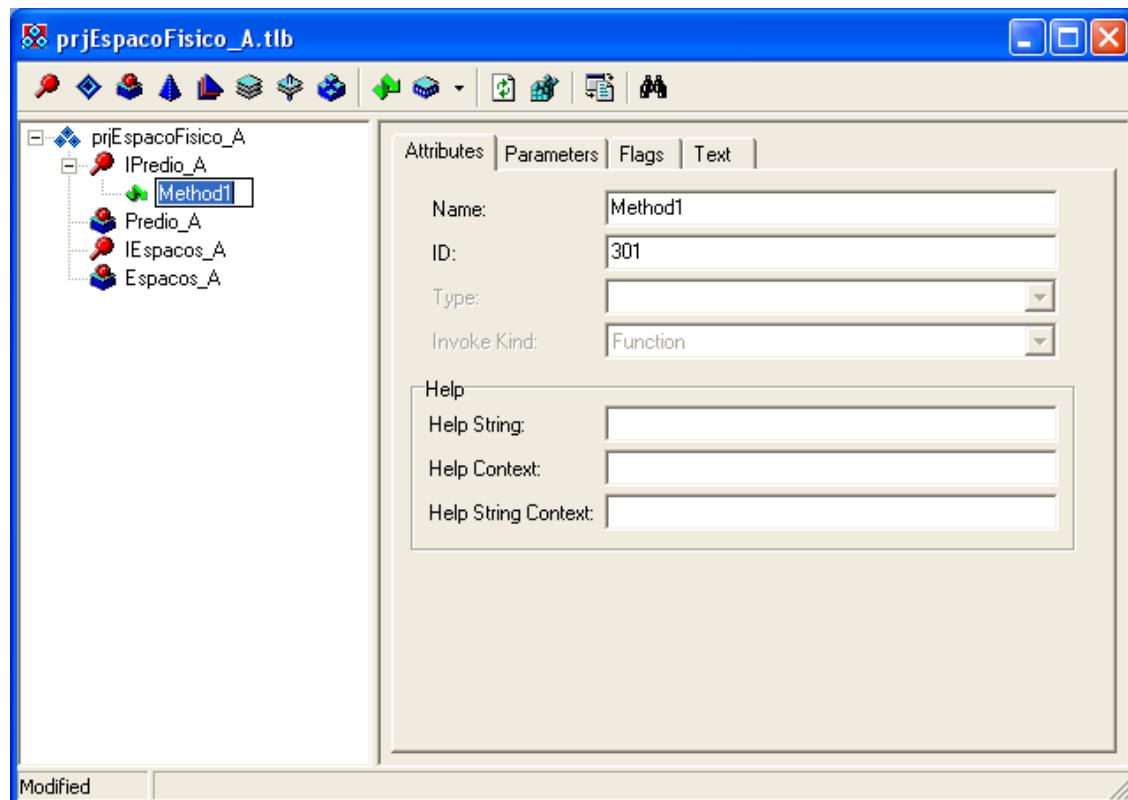
type
  TdmPredio_A = class(TDmBusiness, IPredio_A)
protected
  procedure InitMetaComp(aMetaComp: TBsMetaComponent); override;
  function GetEspacos( IdPredio: Integer; CtxSGCA: OleVariant;
    const CtxDb: IBsContextoDb; var Errors: ICpErrors): OleVariant;
end;

```

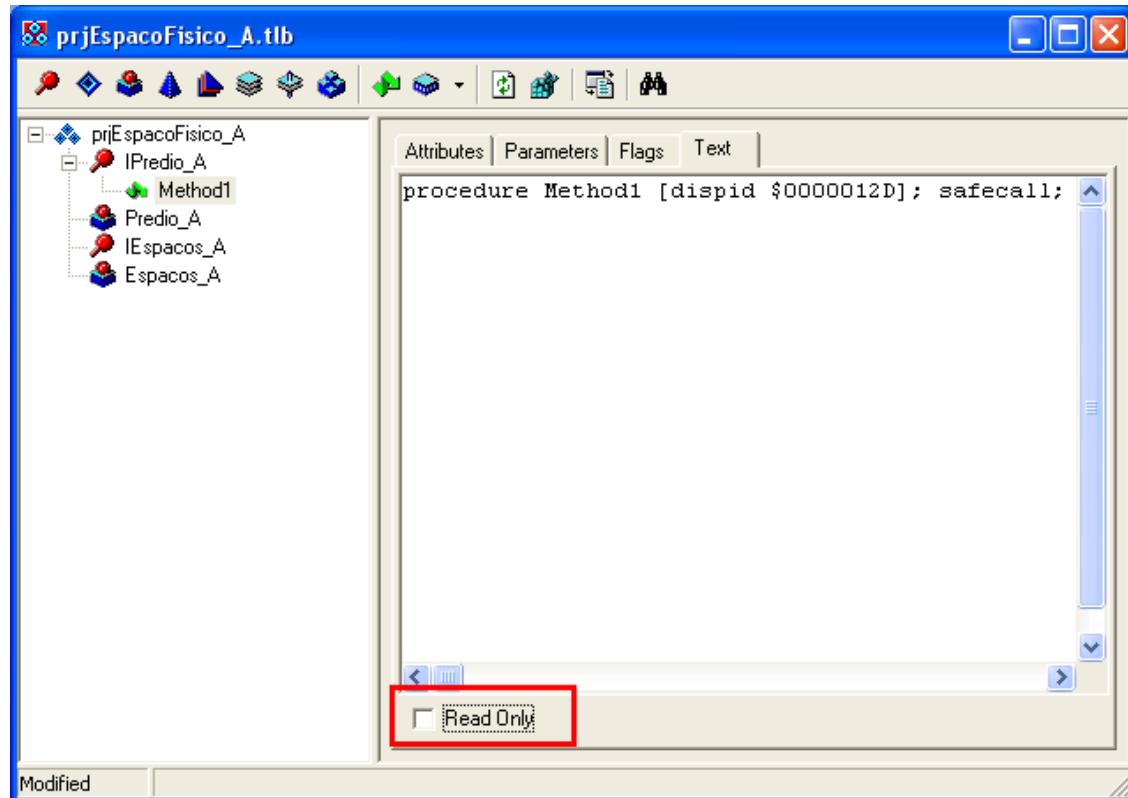
Selecione o código como mostrado na figura, antes de chegar no ; final.

Abra o editor da *Type Library*. View -> Type Library:

Selecione a interface IPredio\_A e clique em :

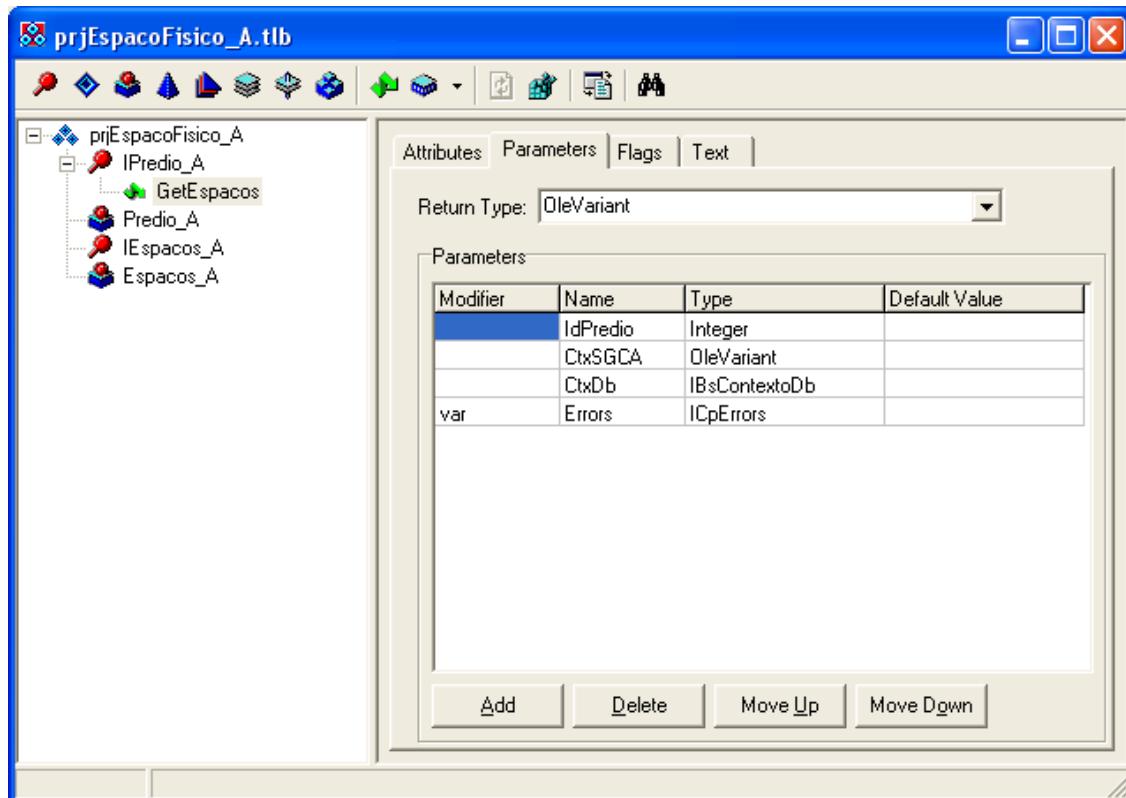


Vá até a última aba Text, e desmarque a opção "Read Only"



Antes de executar o procedimento, certifique-se de que *CpBusiness*, *CpErrors* e *PrjContextoDB* está no uses da *Type Library*, conforme demonstrado na seção de [configuração](#) [47].

Agora cole a assinatura do método copiado por cima de "procedure Method1":



Salve o projeto. Como alteramos a *interface* do objeto COM, precisamos reinstalar o componente no COM+. Para isso recompile a aplicação servidora, abra o Serviços de Componente, exclua os componentes do pacote *pkEspacoFisico\_A* e registre a DLL novamente. Isso é necessária para que o servidor reconheça o novo método. Caso contrário, o erro “*Interface not supported*” ou “*Membro não encontrado*” pode acontecer.

Abra o pacote *pknEspacoFisico.dpk* e recompile-o para que as aplicações clientes possam utilizar o novo método.

### 3.10 Adicionando cadastro de espaços

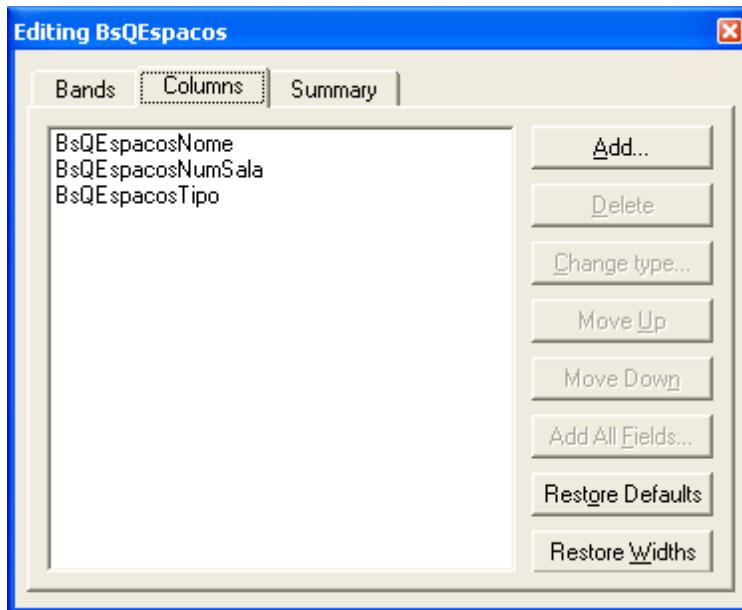
Na segunda aba do TBsPageControl, renomeie a aba para “tbshSalas”. E adicione a segunda linha ao código do evento *OnShow* do formulário:

```
procedure TfrPredios.FormShow(Sender: TObject);
begin
  inherited;
  tbshPredio.Caption := dmPrediosEspaco.CliPredio_A1.DisplayLabelSingular;
  tbshSalas.Caption := dmPrediosEspaco.CliEspacos_A1.DisplayLabelSingular;
end;
```

Coloque na aba *tbshSalas* um *TBsQuantumGrid* (paleta *Bs Additional*) e configure suas propriedades como mostrado a seguir:

```
object BsQEspacos: TBsQuantumGrid
  KeyField = 'ID_ESPACO'
  Align = alClient
  BsClient = dmPrediosEspaco.CliPredio_A1
  BsDataSource = 'dsGetEspacos'
```

Dê um duplo clique no *BsQuantumGrid* e irá aparecer o editor de colunas:



Adicione 3 colunas no botão *Add..* e configure conforme abaixo:

```
object BsQEspacosNome: TdxDBGridColumn
  FieldName = 'NOME_ESPACO'
end
object BsQEspacosNumSala: TdxDBGridColumn
  FieldName = 'NUM_SALA'
end
object BsQEspacosTipo: TdxDBGridColumn
  FieldName = 'TIPO_ESPACO'
end
```

Declare o seguinte método na seção *protected* do formulário:

```
protected  
  ||| procedure CarregaRotulos; override;
```

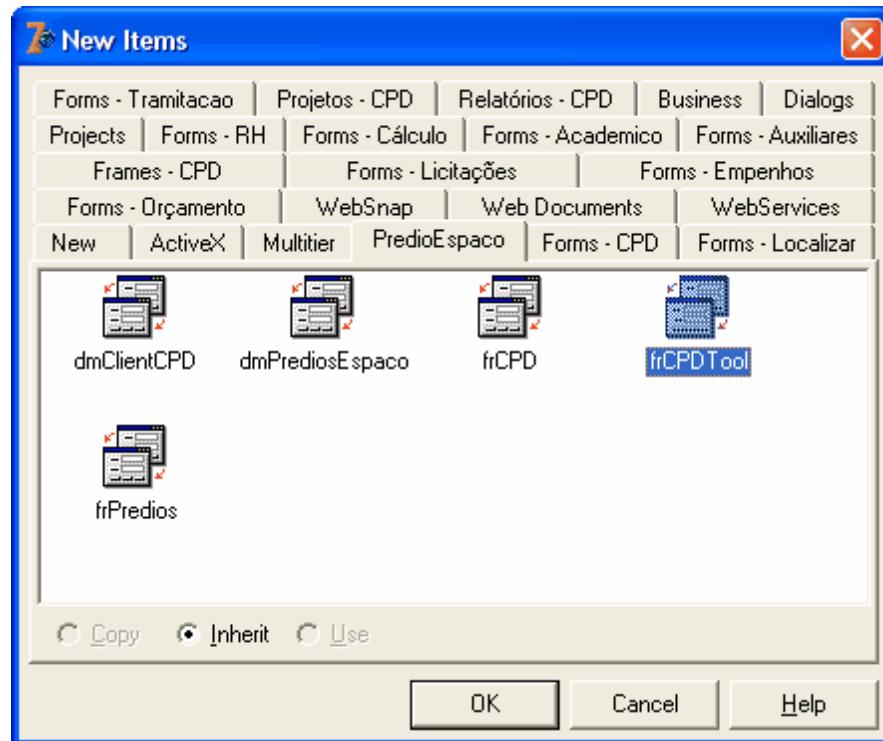
Digite Ctrl + Shift + C para implementar o código:

```
procedure TfrPredios.CarregaRotulos;  
begin  
  inherited;  
  with dmPrediosEspaco.CliEspacos_A1 do  
  begin  
    BsQEspacosNome.Caption := FieldByName('NOME_ESPACO').DisplayLabel;  
    BsQEspacosNumSala.Caption := FieldByName('NUM_SALA').DisplayLabel;  
    BsQEspacosTipo.Caption := FieldByName('TIPO_ESPACO_ITEM').DisplayLabel;  
  end;  
end;
```

No evento *OnChange* do BsPageControl, adicione o seguinte código:

```
procedure TfrPredios.BsPageControl1Change(Sender: TObject);  
begin  
  if BsPageControl1.ActivePage = tbshSalas then  
    with dmPrediosEspaco do  
    begin  
      CliPredio_A1.GetEspacos(CliPredio_A1.FieldByName('ID_PREDIO').AsInteger);  
      Detalhes.Enabled := not dmPrediosEspaco.CliPredio_A1.cdsGetEspacos.IsEmpty;  
      NovoGrid.Enabled := True;  
    end  
  else  
  begin  
    Detalhes.Enabled := False;  
    NovoGrid.Enabled := False;  
  end;  
end;
```

Agora vamos criar o cadastro de espaços. Clique em File -> New -> Other -> PredioEspaco



Selecione *frCPDTool* e dê um OK. Salve o formulário com o nome *ufrEspacos.pas* e renomeie o formulário para *frEspacos*. Aperte Alt+F11 e selecione a unit *udmPrediosEspacos* na lista.

Coloque no formulário sete componentes BsContainer, configurando suas propriedades como mostrado a seguir:

```

object bscNomePredio: TBsContainer
  BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
  BsClientData.FieldName = 'NOME_PREDIO'
  BsControlType = bstEdit
  object bscNomePredioEdit: TBsEdit
    ParentColor = True
    ReadOnly = True
    BsClientData.BsClient = dmPrediosEspaco.CliPredio_A1
    BsClientData.FieldName = 'NOME_PREDIO'
    BsClientData.BsDataSource = 'dsGetRecord'
  end
end
object bscNomeEspaco: TBsContainer
  BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
  BsClientData.FieldName = 'NOME_ESPACO'
  BsControlType = bstEdit
  object bscNomeEspacoEdit: TBsEdit
    BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
    BsClientData.FieldName = 'NOME_ESPACO'
    BsClientData.BsDataSource = 'dsGetRecord'
  end
end
object bscArea: TBsContainer
  BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
  BsClientData.FieldName = 'AREA'

```

```
BsControlType = bstEdit
object bscAreaEdit: TBsEdit
  BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
  BsClientData.FieldName = 'AREA'
  BsClientData.BsDataSource = 'dsGetRecord'
end
end
object bscDimensao: TBsContainer
  BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
  BsClientData.FieldName = 'DIMENSOES'
  BsControlType = bstEdit
  object bscDimensaoEdit: TBsEdit
    BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
    BsClientData.FieldName = 'DIMENSOES'
    BsClientData.BsDataSource = 'dsGetRecord'
  end
end
object bscCapacidade: TBsContainer
  BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
  BsClientData.FieldName = 'CAPACIDADE'
  BsControlType = bstEdit
  object bscCapacidadeEdit: TBsEdit
    BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
    BsClientData.FieldName = 'CAPACIDADE'
    BsClientData.BsDataSource = 'dsGetRecord'
  end
end
object bscTipoEspaco: TBsContainer
  BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
  BsClientData.FieldName = 'TIPO_ESPACO_ITEM'
  BsControlType = bstTabEstrut
  object bscTipoEspacoTabEstrut: TBsTabEstrut
    BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
    BsClientData.FieldName = 'TIPO_ESPACO_ITEM'
  end
end
object bscNumSala: TBsContainer
  BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
  BsClientData.FieldName = 'NUM_SALA'
  BsControlType = bstEdit
  object bscNumSalaEdit: TBsEdit
    BsClientData.BsClient = dmPrediosEspaco.CliEspacos_A1
    BsClientData.FieldName = 'NUM_SALA'
    BsClientData.BsDataSource = 'dsGetRecord'
  end
end
end
```



No evento *OnCreate* do formulário, adicione o seguinte código:

```
procedure TfrEspacos.FormCreate(Sender: TObject);
begin
  inherited;
  CliPrincipal := dmPrediosEspaco.CliEspacos_A1;
  FNomeExecutavel := 'EFMPredios.exe';
  AddDataModule(dmPrediosEspaco);
end;
```

No botão Novo, adicione o seguinte código:

```
procedure TfrEspacos.NovoExecute(Sender: TObject);
begin
  inherited;
  dmPrediosEspaco.CliEspacos_A1.Append(True);
end;
```

No botão Alterar:

```
procedure TfrEspacos.AlterarExecute(Sender: TObject);
begin
  inherited;
  dmPrediosEspaco.CliEspacos_A1.Edit;
end;
```

No botão Excluir:

```

procedure TfrEspacos.ExcluirExecute(Sender: TObject);
const CONFIRMA_EXCLUSAO = 2;
begin
  with dmPrediosEspaco do
    if (MessageDlg(CONFIRMA_EXCLUSAO, mtcConfirmacao, [mbcSim,mbcNao],
      0, '') , [CliEspacos_A1.FieldByName('NOME_ESPACO').Value]) = mrYes) then
    begin
      AtualizacaoEfetuada := CliEspacos_A1.Delete;
      if AtualizacaoEfetuada then
        inherited;
    end;
end;

```

Declare o seguinte método na seção *protected* do formulário:

```

protected
  procedure EnableDisableControls(aEnable : Boolean); override;

```

Digite Ctrl + Shift + C para implementar o código:

```

procedure TfrEspacos.EnableDisableControls(aEnable: Boolean);
var i: Integer;
begin
  inherited;
  for i:=0 to ComponentCount-1 do
    if Components[i] is TBsContainer then
      (Components[i] as TBsContainer).Enabled := aEnable;
end;

```

No botão Salvar adicione o código:

```

procedure TfrEspacos.SalvarExecute(Sender: TObject);
begin
  with dmPrediosEspaco do
  begin
    CliEspacos_A1.FieldByName('ID_PREDIO').Value :=
      CliPredio_A1.FieldByName('ID_PREDIO').Value;
    case frmState of
      fstInsert : AtualizacaoEfetuada := CliEspacos_A1.Insert;
      fstEdit : AtualizacaoEfetuada := CliEspacos_A1.Update;
    end;
    if AtualizacaoEfetuada then
      inherited;
  end;
end;

```

No botão Cancelar:

```
procedure TfrEspacos.CancelarExecute(Sender: TObject);
begin
  dmPrediosEspaco.CliEspacos_A1.Cancel;
  inherited;
end;
```

Declara e implemente o método *CloseDataSets* (protected), com **override**:

```
protected
  procedure CloseDataSets; override;
```

Implemente da seguinte forma:

```
procedure TfrEspacos.CloseDataSets;
begin
  inherited;
  with dmPrediosEspaco do
    if not ShowingAsNew then
      begin
        CliEspacos_A1.CliDsGetRecord.Close;
        CliPredio_A1.CliDsGetRecord.Close;
      end;
end;
```

Declare os seguintes métodos na seção **protected** da classe do formulário:

```
procedure ShowAsNone(ForeignValue: Integer; Reserved: Variant); override;
procedure ShowAsNew(ForeignValue: integer; Reserved: Variant); override;
```

```
procedure TfrEspacos.ShowAsNew(ForeignValue: integer; Reserved: Variant);
begin
  dmPrediosEspaco.CliPredio_A1.GetRecordPK(ForeignValue, True, True);
  NovoExecute(Self);
end;
```

```
procedure TfrEspacos.ShowAsNone(ForeignValue: Integer; Reserved: Variant);
begin
  if dmPrediosEspaco.CliEspacos_A1.GetRecordPK(ForeignValue, True, True) then
    PosicionaRegistro;
end;
```

Para maiores detalhes, ver no [Apêndice B](#)<sup>[150]</sup>, [ShowAsNone](#)<sup>[152]</sup> e [ShowAsNew](#)<sup>[152]</sup>.

Volte ao formulário principal (ffPredios), dê um duplo clique no *ActionList actlPrincipal* , selecione as ações *NovoGrid* e *Detalhes* e altere a propriedade *Visible* para True.



No evento *OnExecute* da ação *NovoGrid* digite:

```

procedure TfrPredios.NovoGridExecute(Sender: TObject);
begin
  inherited;
  with dmPrediosEspaco do
    begin
      frEspacos.ShowAsAplicacaoChamada(0, tstNew, 0,
        CliPredio_A1.PKField.Value, Null);
      if frEspacos.AtualizacaoEfetuada then
        CliPredio_A1.GetEspacos(CliPredio_A1.PKField.AsInteger);
    end;
end;

```

OBS: *PKField.Value* recupera o valor do campo da chave física da tabela, que está em no registro corrente.

No evento *OnExecute* da ação *Detalhes* digite:

```
procedure TfrPredios.DetalhesExecute(Sender: TObject);
begin
  inherited;
  with dmPrediosEspaco do
  begin
    frEspacos.ShowAsAplicacaoChamada(1, tstNone, Null,
      CliPredio_A1.cdsGetEspacos.FieldByName('ID_ESPACO').AsInteger, Null);
    if frEspacos.AtualizacaoEfetuada then
      CliPredio_A1.GetEspacos(CliPredio_A1.PKField.AsInteger);
  end;
end;
```

A aplicação cliente para dar manutenção nas tabelas PREDIOS\_A e ESPACOS\_A está pronta para testes! Antes de disponibilizar a mesma no ambiente de produção, não esqueça de colocar de volta a linha "if *UsuarioAutorizado* ....", para evitar que a aplicação possa ser aberta por fora do GCA Navegacao.

# **Top Level Intro**

This page is printed before a new  
top-level chapter starts

**Part**



**IV**

## 4 Alterando a servidora

Até o presente momento, utilizamos métodos prontos de *TDmBusiness* para inserir, atualizar e excluir dados. Na aplicação cliente, utilizamos um formulário padrão, que já possui barras de ferramentas (novo, alterar, excluir, salvar, cancelar), além de controles customizados para serem utilizados em conjunto com os *TCliBusiness*.

Só foi preciso criar um método novo para retornar todos os "espaços" cadastrados em um prédio. Algumas vezes precisamos efetuar mais operações antes de inserir ou atualizar os dados, ou até mesmo efetuar novas consistências além das efetuadas nos métodos *ConsisteInsert*, *ConsisteUpdate* e *ConsisteDelete*.

### 4.1 Redefinindo o Update

Na tabela PREDIOS\_A, existe um campo chamado OBSERVACAO. Ao alterar um prédio, deseja-se inserir na observação o nome do usuário que fez a última alteração, assim como uma listagem contendo todos os números das salas daquele prédio, por exemplo a observação ficaria da seguinte forma:

```
** Última Alteração: 21/09/2010 10:18:02 - [ADM] Usuário Administrador
| Salas cadastradas para o Prédio X:
| XXX, YYY, ZZZ
```

Além disso, deseja-se mostrar um mensagem na tela, informando que o método foi executado com sucesso.

A aplicação cliente continua da mesma forma, ou seja, chamando o método *Update* do *TCliPredio\_A*. Portanto, vamos sobreescrver (*override*) o método *Update* de *TDmBusiness* para adicionar mais essa informação antes de efetivamente atualizar os dados.

Abra a servidora *prjEspacoFisico\_A* e depois a *udmPredio\_A*. Adicione na clausula **uses** as seguintes units: **DBClient**, **prjGCA\_TLB**, **uBsException**, **UtilCPDSrv**.

Declare o seguinte método na seção **protected**:

```
procedure Update(Values: OleVariant; out ConcorGerado: integer; CtxSGCA: OleVariant;
const CtxDb: IBsContextoDb; var Errors: ICpErrors); override;
```

Digite Ctrl + Shift + C para implementar o código:

```
procedure TdmPredio_A.Update(Values: OleVariant; out ConcorGerado: integer;
  CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors);
var
  aCtx: IBsContextoDb;
begin
  try try
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    inherited Update(Values, ConcorGerado, CtxSGCA, aCtx, Errors);
  except
    on E: Exception do
      GetCpErrorsParam(E, Errors, 176, Name, ['Update'], CtxSGCA);
  end;
  finally
    SetComplete;
  end;
end;
```

Esse é o código básico da servidora. Nesse momento, o código está exatamente igual ao *Update* de *TDmBusiness*, pois o método é iniciado e somente é feita uma chamada ao método implementado em *TDmBusiness* (através do comando **inherited**). Ou seja, nada de novo está sendo feito além do que já está implementado em *TDmBusiness*.

Para iniciar as alterações, será preciso as seguintes informações:

- Lista dos espaços cadastrados para o prédio que está sendo feita a alteração (pode-se utilizar o método *GetEspacos*, já desenvolvido).
- Para usar o método *GetEspacos*, é necessário saber o ID\_PREDIO.

Implemente o código a seguir:

```

procedure TdmPredio_A.Update(Values: OleVariant; out ConcorGerado: integer;
  CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors);
var
  aCtx: IBsContextoDb;
  cdsEspacos: TClientDataSet; 1
  idPredio: Integer;
begin
  try try
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    cdsEspacos := TClientDataSet.Create(nil); 2
    try
      3 idPredio := Values[IndexInFields('ID_PREDIO', CtxSGCA, aCtx, Errors)];
      cdsEspacos.Data := GetEspacos(idPredio, CtxSGCA, aCtx, Errors);

      inherited Update(Values, ConcorGerado, CtxSGCA, aCtx, Errors);
    finally
      cdsEspacos.Free; 2.1
    end;
  except
    on E: Exception do
      GetCpErrorsParam(E, Errors, 176, Name, ['Update'], CtxSGCA);
  end;
  finally
    SetComplete;
  end;
end;

```

1. Criação de duas variáveis: cdsEspacos, que será o *TClientDataSet* que irá armazenar os espaços retornados pelo método GetEspacos. IdPredio para guardar a chave primária do prédio que está sendo modificado.
2. Antes de utilizar o *TClientDataSet*, é preciso criar uma instância do mesmo em memória. Em aplicações servidoras o processo de criação e liberação de objetos deve ser explícito. Por essa razão, no item 2 o objeto é criado e no item 2.1 o objeto é liberado da memória. É aconselhável utilizar um bloco *try ... finally* entre a criação e a liberação de objetos, a fim de garantir que os mesmos serão liberados, mesmo que ocorra algum erro na execução do método.
3. Na primeira linha é buscado o valor correspondente a variável ID\_PREDIO. Todos os valores que foram digitados na aplicação cliente estão encapsulados dentro do *OleVariant* Values. Values nada mais é do que um vetor, de N posições, sendo N o total de colunas físicas da tabela. Quem controla a posição que cada coluna ocupa no vetor é o catálogo do SIM, através do cadastro de componentes. Dessa forma, o programador não necessita decorar essas posições e sim utilizar o método *IndexInFields* (descrito na seção [TDmBusiness](#)<sup>[17]</sup>), que retornará a posição que a coluna está no cadastro de componentes.

Na segunda linha, é executado o mesmo método GetEspacos que foi utilizado na aplicação cliente. A única diferença que é dentro da servidora, o programador que deve criar um *TClientDataSet* para receber o retorno do método e atribuir o retorno na propriedade "Data" do *TClientDataSet*.

Após essa alteração, ainda precisamos dos dados do usuário que está efetuando a alteração (Nome e Login) e a partir desses dados, montar o cabeçalho da observação:

```

procedure TdmPredio_A.Update(Values: OleVariant; out ConcorGerado: integer;
  CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors);
var
  aCtx: IBsContextoDb;
  aUsuario: IUsuarioDisp; 1
  cdsEspacos, cdsUsuario: TClientDataSet; 2
  idPredio, idUsuario: Integer;
  Observacao: String;
begin
  try try
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    aUsuario := GetBsComObj(CLASS_Usuario) as IUsuarioDisp; 1.1
    cdsEspacos := TClientDataSet.Create(nil);
    cdsUsuario := TClientDataSet.Create(nil); 2.1
  try
    idPredio := Values[IndexInFields('ID_PREDIO', CtxSGCA, aCtx, Errors)];
    cdsEspacos.Data := GetEspacos(idPredio, CtxSGCA, aCtx, Errors);
    idUsuario := GetCtxIdUsuario(CtxSGCA);
    cdsUsuario.Data := aUsuario.GetRecordPK(idUsuario, True, CtxSGCA, aCtx, Errors);
    Observacao := '*** Última Alteração: ' + GetServerDateTimeStr +
      ' - [' + cdsUsuario.FieldName('LOGIN').AsString +
      '] ' + cdsUsuario.FieldName('NOME_USUARIO').AsString;
  3
  inherited Update(Values, ConcorGerado, CtxSGCA, aCtx, Errors);
  finally
    cdsEspacos.Free;
    cdsUsuario.Free; 2.2
  end;
  except

```

1. Para buscar os dados relativos ao usuário que está efetuando a alteração, vamos utilizar outra interface COM+, *IUsuario*, que está dentro de *prjGCA.dll*. No item 1, está sendo declarada uma variável *aUsuario* do tipo *IUsuarioDisp*. Na linha 1.1 está sendo instanciada essa interface. Esse é o método padrão para instanciar interfaces que herdam de *ICpBusiness*. Sempre é necessário instanciar uma interface antes de utilizar os seus métodos, caso contrário uma mensagem de *access violation* irá ocorrer. Não é necessário colocar código para liberar a interface, assim como é necessário nos objetos. Ao compilar a servidora, o Delphi adicionará o código necessário para a liberação.
2. Será preciso criar outro *TClientDataSet*, para guardar os dados do usuário. Para isso, deve-se declarar uma variável, instanciar o objeto e após o uso fazer a liberação, como monstra a sequencia 2; 2.1 e 2.2.
3. Como explicado na seções anteriores, *CtxSGCA*<sup>[146]</sup> é o pacote de segurança do sistema. Ele é um vetor com 14 posições com muitas informações importantes, entre elas o identificador do usuário que está executando o método. Da mesma forma que acontece na variável *Values*, não é necessário decorar a posição do vetor que está o valor do usuário. Pode-se utilizar o método *GetCtxIdUsuario* da unit *UtilCPDSrv*.

Na segunda linha, é utilizado o método *GetRecordPK* da interface *IUsuario*, para retornar todas as informações do usuário passado por parâmetro.

Na terça linha, é montado o cabeçalho fixo da observação, com as informações do usuário e da data e hora de alteração.

Após essas alterações, basta utilizar os espaços retornados pelo método *GetEspacos* e montar o resto da observação. Ao final, grava-se o novo valor no campo *OBSERVACAO*:

```

Observacao := '*** Última Alteração: ' + GetServerDateTimeStr +
  ' - [' + cdsUsuario.FieldByName('LOGIN').AsString +
  '] ' + cdsUsuario.FieldByName('NOME_USUARIO').AsString;

if cdsEspacos.RecordCount > 0 then
begin
  Observacao := Observacao + #13#10 + 'Salas cadastradas para o Prédio ' +
  + Values[IndexInFields('NOME_PREDIO', CtxSGCA, aCtx, Errors)] + ':' + #13#10;
  while not cdsEspacos.Eof do
  begin
    Observacao := Observacao +
      Trim(cdsEspacos.FieldByName('NUM_SALA').AsString) + ', ';
    cdsEspacos.Next;
  end;
end
else
  Observacao := Observacao + #13#10 + 'Nenhuma sala cadastrada ' +
  'para o Prédio ' +
  Values[IndexInFields('NOME_PREDIO', CtxSGCA, aCtx, Errors)];

Values[IndexInFields('OBSERVACAO', CtxSGCA, aCtx, Errors)] := Observacao;

inherited Update(Values, ConcorGerado, CtxSGCA, aCtx, Errors);

```

Nessa parte do método, somente é percorrido o *TClientDataSet* e a informação do número da sala é concatenada na observação. Caso nenhum espaço tenha sido retornado no método, a mensagem "Nenhuma sala cadastrada" será gravada. Na última linha, o novo valor da observação é adicionada no vetor *Values*.

Falta somente adicionar a mensagem informando que o método foi executado com sucesso. Vamos adicionar uma mensagem do tipo *Information* dentro da variável *Errors*:

```

    Values[IndexInFields('OBSERVACAO', CtxSGCA, aCtx, Errors)] := Observacao;

    inherited Update(Values, ConcorGerado, CtxSGCA, aCtx, Errors);

    if SemErro(Errors) then
        GetCpErrorsParam(EBsInformation.Create('Dados atualizados com sucesso!',
                                              CtxSGCA, aCtx), Errors, 176, Name, ['Update'], CtxSGCA);
    finally
        cdsEspacos.Free;
        cdsUsuario.Free;
    end;

```

Abaixo o código completo do método:

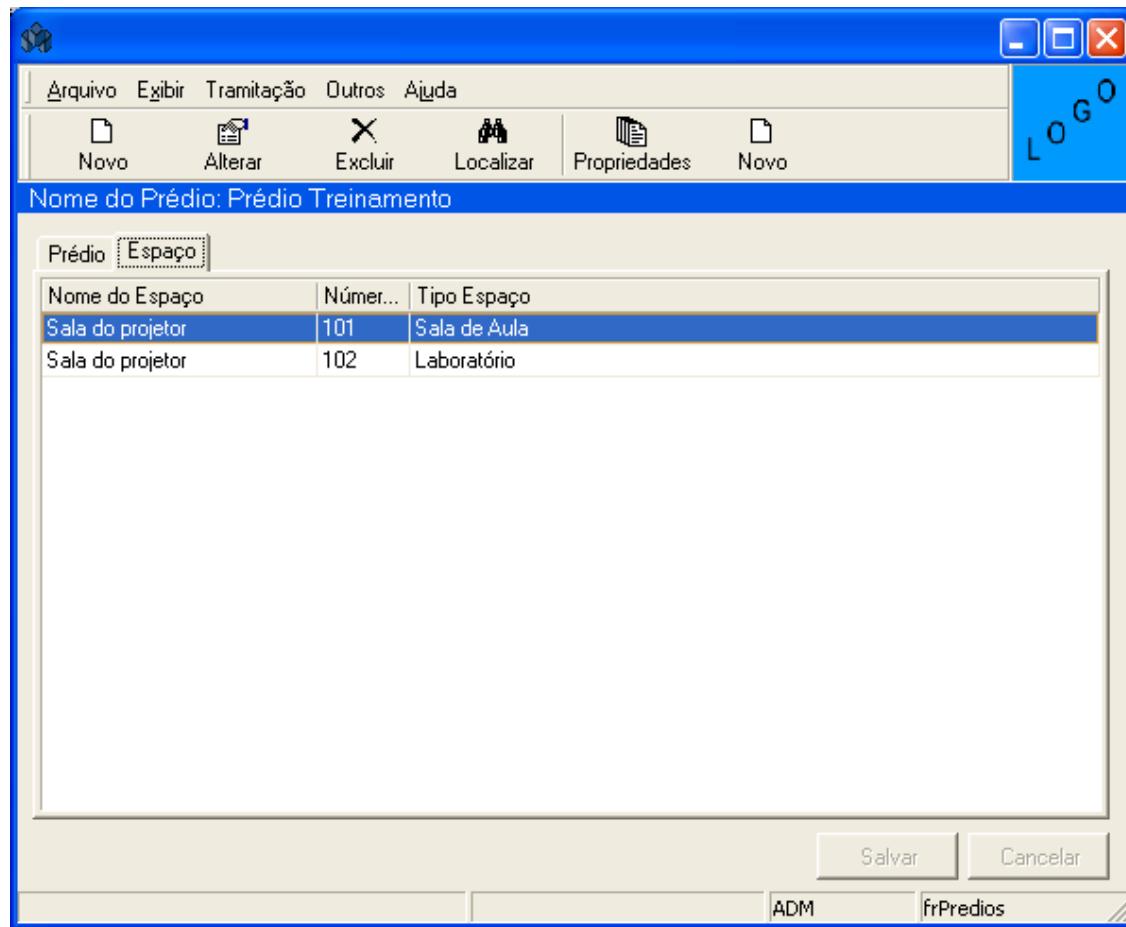
```

procedure TdmPredio_A.Update(Values: OleVariant; out ConcorGerado: integer;
                            CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors);
var
    aCtx: IBsContextoDb;
    cdsEspacos, cdsUsuario: TClientDataSet;
    aUsuario: IUsuarioDisp;
    idUsuario, idPredio: Integer;
    Observacao: String;
begin
    try try
        aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
        aUsuario := GetBsComObj(CLASS_Usuario) as IUsuarioDisp;
        cdsEspacos := TClientDataSet.Create(nil);
        cdsUsuario := TClientDataSet.Create(nil);
        try
            idUsuario := GetCtxIdUsuario(CtxSGCA);
            idPredio := Values[IndexInFields('ID_PREDIO', CtxSGCA, aCtx, Errors)];
            cdsUsuario.Data := aUsuario.GetRecordPK(idUsuario, True, CtxSGCA, aCtx,
Errors);
            cdsEspacos.Data := GetEspacos(idPredio, CtxSGCA, aCtx, Errors);
            Observacao := '** Última Alteração: ' + GetServerDateTimeStr +
                          ' - [' + cdsUsuario.FieldName('LOGIN').AsString +
                          '] ' + cdsUsuario.FieldName('NOME_USUARIO').AsString;
            if cdsEspacos.RecordCount > 0 then
                begin
                    Observacao := Observacao + #13#10 + 'Salas cadastradas para o Prédio ' +
                                  Values[IndexInFields('NOME_PREDIO', CtxSGCA, aCtx, Errors)] + ':' +
#13#10;
                    while not cdsEspacos.Eof do
                        begin
                            Observacao := Observacao +
                                         Trim(cdsEspacos.FieldName('NUM_SALA').AsString) + ', ';
                            cdsEspacos.Next;
                        end;
                end
            else
                Observacao := Observacao + #13#10 + 'Nenhuma sala cadastrada ' +
                              'para o Prédio ' +
                              Values[IndexInFields('NOME_PREDIO', CtxSGCA, aCtx, Errors)];
            Values[IndexInFields('OBSERVACAO', CtxSGCA, aCtx, Errors)] := Observacao;
            inherited Update(Values, ConcorGerado, CtxSGCA, aCtx, Errors);
    end;

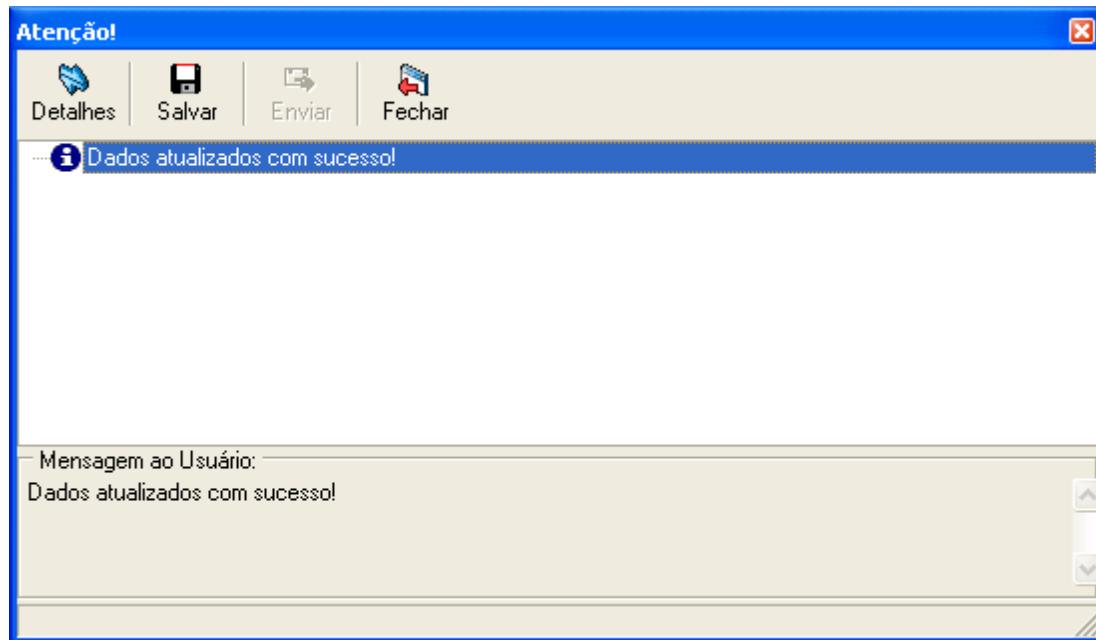
```

```
if SemErro(Errors) then
    GetCpErrorsParam(EBsInformation.Create('Dados atualizados com sucesso!',
                                             CtxSGCA, aCtx), Errors, 176, Name, ['Update'], CtxSGCA);
finally
    cdsEspacos.Free;
    cdsUsuario.Free;
end;
except
    on E: Exception do
        GetCpErrorsParam(E, Errors, 176, Name, ['Update'], CtxSGCA);
    end;
finally
    SetComplete;
end;
end;
```

Volte para a aplicação cliente. Localize algum prédio cadastrado, e verifique na aba "Espaço", se o mesmo possui mais de uma sala cadastrada, como no exemplo abaixo:



Clique no botão Alterar, e depois em Salvar, para executar o método Update:



Veja como ficou a observação:



## 4.2 Exemplo de transação

Nesse outro exemplo, o método Insert que será sobreescrito. No momento que o usuário estiver cadastrando um prédio novo, deseja-se que um novo espaço também seja adicionado, com as seguintes informações para as colunas:

- NOME\_ESPACO = "Espaço 1"
- TIPO\_ESPACO\_TAB = 102
- TIPO\_ESPACO\_ITEM = 1
- NUM\_SALA = "1"
- ID\_PREDIO = (Usar o novo que foi incluído no próprio Insert)

Declare o seguinte método na seção protected:

```
procedure Insert(Values: OleVariant; out AutoIncGerado: integer; CtxSGCA: OleVariant;
const CtxDb: IBsContextoDb; var Errors: ICpErrors); override;
```

Digite Ctrl + Shift + C para implementar o código:

```

procedure TdmPredio_A.Insert(Values: OleVariant; out AutoIncGerado: integer;
  CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors);
var aCtx: IBsContextoDb;
  aEspacosA: IEspacos_ADisp;
begin
  try try
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    aEspacosA := GetBsComObj(CLASS_Espacos_A) as IEspacos_ADisp;

    inherited Insert(Values, AutoIncGerado, CtxSGCA, aCtx, Errors);
  except
    on E: Exception do
      GetCpErrorsParam(E, Errors, 176, Name, ['Insert'], CtxSGCA);
  end;
  finally
    SetComplete;
  end;
end;

```

Esse código somente está instanciando uma interface do tipo IEspacos\_A, pois será necessária para inserir o espaço novo, juntamente com o prédio. Além disso, será preciso criar uma transação, pois se acontecer algum erro durante a criação do espaços, o procedimento deve ser desfeito, assim com o prédio que já havia sido inserido. O código padrão para utilização de transação está demonstrando abaixo:

```

procedure TdmPredio_A.Insert(Values: OleVariant; out AutoIncGerado: integer;
  CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors);
var aCtx: IBsContextoDb;
  aEspacosA: IEspacos_ADisp;
begin
  try try
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    aEspacosA := GetBsComObj(CLASS_Espacos_A) as IEspacos_ADisp;
    aCtx.InitDbTrans(CtxSGCA, Errors);
    try
      inherited Insert(Values, AutoIncGerado, CtxSGCA, aCtx, Errors);

      if SemErro(Errors) then
        aCtx.CommitDbTrans(CtxSGCA, Errors)
      else
        aCtx.RollbackDbTrans(CtxSGCA, Errors);
    except
      aCtx.RollbackDbTrans(CtxSGCA, Errors);
      raise;
    end;
  except
    on E: Exception do
      GetCpErrorsParam(E, Errors, 176, Name, ['Insert'], CtxSGCA);
  end;
end;

```

O comando para iniciar uma transação é o comando *aCtx.InitDbTrans*. Imediatamente abaixo desse código, deve haver um bloco *try ... except*, para garantir que a transação será desfeita (*RollBack*) caso ocorra algum erro durante a execução do método. Esse é modelo de código para uso de transação. Todo o código que ficará dentro da transação deve estar abaixo do *try*. Nesse exemplo, está sendo feita uma chamada ao Insert da *TDmBusiness*, para gravar o prédio. Isso está correto, porém, após inserir o prédio, devemos criar o novo espaço:

```

procedure TdmPredio_A.Insert(Values: OleVariant; out AutoIncGerado: integer;
  CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors);
var aCtx: IBsContextoDb;
  aEspacosA: IEspacos_ADisp;
  ValuesEspaco: OleVariant;
  idEspacoGerado: Integer;
begin
  try try
    aCtx := ConsisteBsContextoDb(CtxSGCA, CtxDb, Errors);
    aEspacosA := GetBsComObj(CLASS_Espacos_A) as IEspacos_ADisp;
    aCtx.InitDbTrans(CtxSGCA, Errors);
    try
      inherited Insert(Values, AutoIncGerado, CtxSGCA, aCtx, Errors);

      if SemErro(Errors) then
        begin
          ValuesEspaco := VarArrayCreate([0, aEspacosA.FieldsCount(CtxSGCA, aCtx, Errors)-1], varVariant);
          ValuesEspaco[aEspacosA.IndexInFields('NOME_ESPAÇO', CtxSGCA, aCtx, Errors)] := 'Espaço 1';
          ValuesEspaco[aEspacosA.IndexInFields('TIPO_ESPAÇO_TAB', CtxSGCA, aCtx, Errors)] := 102;
          ValuesEspaco[aEspacosA.IndexInFields('TIPO_ESPAÇO_ITEM', CtxSGCA, aCtx, Errors)] := 1;
          ValuesEspaco[aEspacosA.IndexInFields('NUM_SALA', CtxSGCA, aCtx, Errors)] := '1';
          ValuesEspaco[aEspacosA.IndexInFields('ID_PREDIO', CtxSGCA, aCtx, Errors)] := AutoIncGerado;
          aEspacosA.Insert(ValuesEspaco, idEspacoGerado, CtxSGCA, aCtx, Errors);
        end;
    end;
  end;
end;

```

Nesse trecho de código, a variável *ValuesEspaco* do tipo *OleVariant* é criada com N posições, sendo N o total de colunas físicas existentes. Novamente não é necessário decorar quantas colunas existem na tabela, ou alterar caso alguma coluna seja adicionada. Basta utilizar o método *FieldsCount*, disponível em todas as interfaces que herdam de *ICpBusiness*, que o total de colunas cadastradas no componente será retornado.

Após, *ValuesEspaco* é preenchida com os valores necessários. Deve-se utilizar o método *IndexInFields* da interface *IEspacosA* para retornar em que posição do vetor se encontra a coluna que deseja-se atribuir os valores.

Finalmente, o método *Insert* de *IEspacosA* é chamado para criar o novo espaço no SGBD. Caso ocorra algum erro, toda a transação será desfeita com o comando *RollBack*.

Abra a aplicação cliente e insira um novo prédio. Se inserir com sucesso, clique na segunda aba e verifique se o novo espaço também foi incluído.

# **Top Level Intro**

This page is printed before a new  
top-level chapter starts

**Part**



V

## 5 Criando Relatórios no SI\*

Caracteriza-se pela elaboração rápida de relatórios, utiliza o editor de formulários do próprio Delphi e baseia-se no fato de que a maior parte dos relatórios é composta por faixas de informações, onde são colocados componentes visíveis, como campo de banco da dados, rótulos, imagens e outros componentes imprimíveis.

- É o controlador do relatório
- Transforma o formulário do Delphi em um relatório
- É ativado pelos métodos Print e Preview.
- Permite visualizar o relatório em tempo de desenvolvimento ao se selecionar a opção Preview .

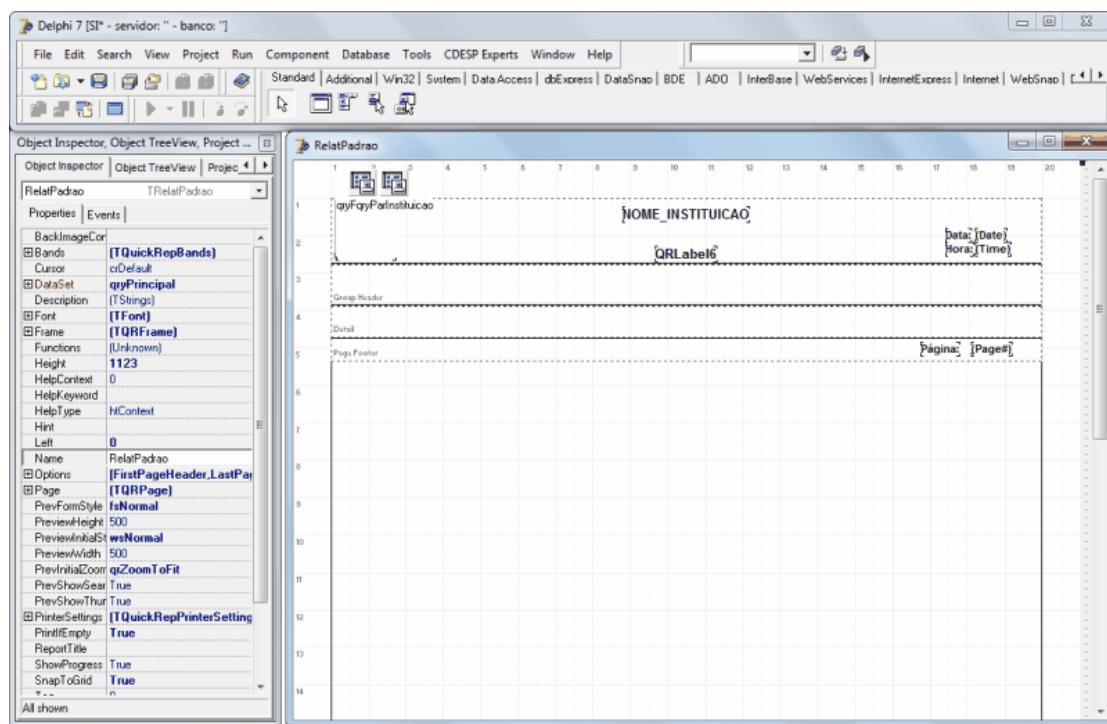


### 5.1 QuickReport

O QuickReport é um ambiente de desenvolvimento rápida de relatórios. Utiliza o editor de formulários do próprio Delphi e baseia-se no fato de que a maior parte dos relatórios é composta por faixas de informações, onde são colocados componentes visíveis, como campo de banco da dados, rótulos, imagens e outros componentes imprimíveis. Ele controla o relatório, o seu leiaute e visualização. Tem componentes próprios para a sua criação, mas permite uma grande personalização através do ambiente Delphi.

Os relatórios criados em QuickReport tem normalmente o formato "tabelar", como os usado em planilhas, podendo ter cabeçalho, agrupamento, somatório e etc.

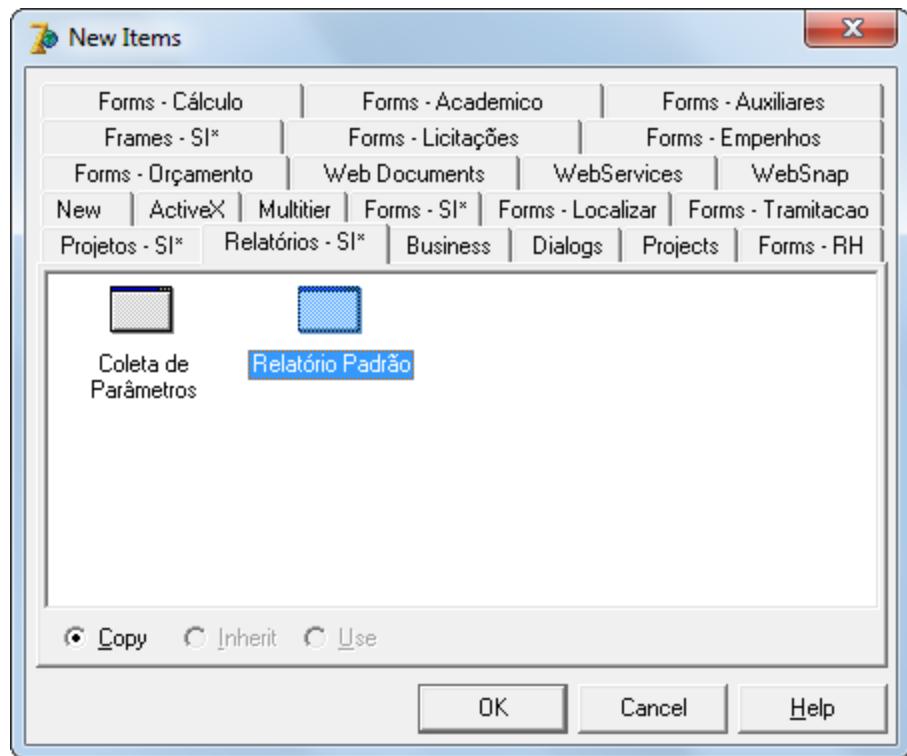
Um exemplo de relatório criado usando QuickReport pode ser visto abaixo:



### 5.1.1 Leiaute

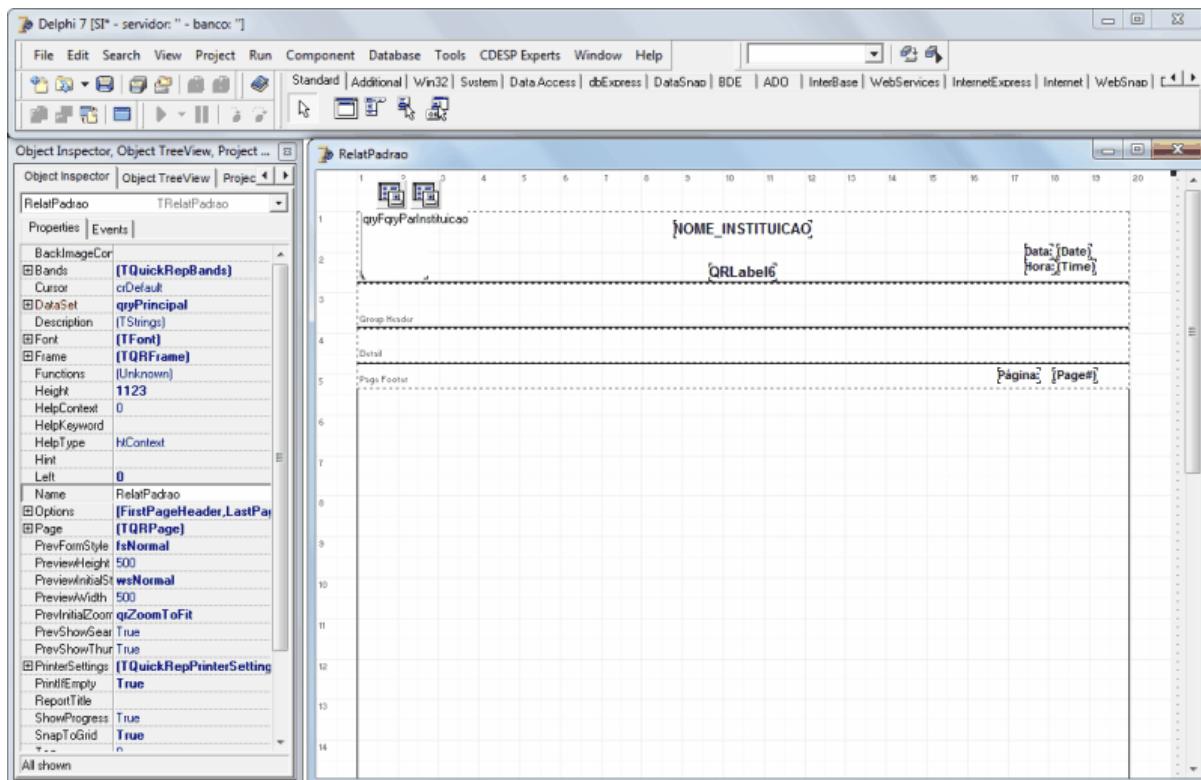
#### Novo Relatório:

O relatório em QuickReport é desenvolvido em ambiente Delphi. Para iniciar um novo relatório, deve-se proceder os seguintes passos: Dentro do Delphi, vá no menu "File/New/Other", onde será aberto a janela para escolher o tipo de item, conforme imagem abaixo. Selecione a aba "Relatórios - SI\*". Será mostrado dois itens. "Coleta de Parâmetros" e "Relatório Padrão". Selecione "Relatório Padrão" e clique no botão "OK".



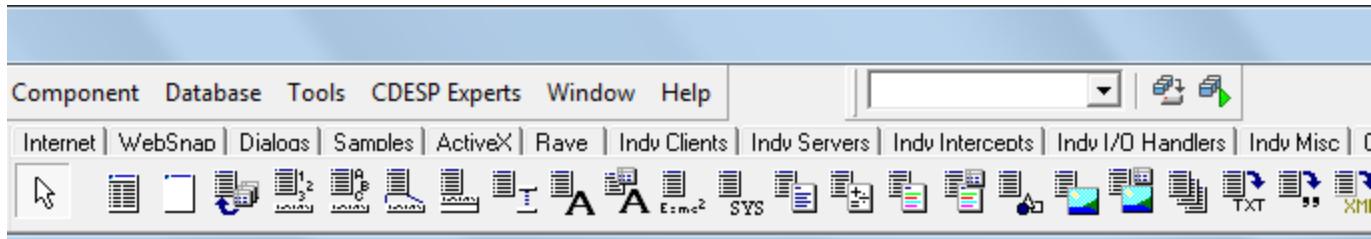
Novo relatório

Após clicar em "OK", será mostrado a seguinte tela:



### Relatório padrão do SI\*

E selecionando "QReport" entre as abas de componentes do Delphi, podemos ver os seguintes componentes para uso no relatório:



**Componentes para criação de relatórios.**

#### **Leiaute:**

O leiaute do relatório funciona baseado em "faixas" (Bands). Faixas são áreas especiais do relatório. Dependendo do tipo da faixa, o funcionamento do relatório é alterado, gerando assim um leiaute diferente. Existem, por exemplo, faixas para cabeçalho, rodapé, agrupamentos, sumário e etc.

O relatório padrão (imagem acima) aparece com algumas faixas já colocadas. Podem ser adicionadas faixas adicionais, ou alterado as existentes.

**OBS:** A função principal das faixas é organizar o leiaute. Faixas diferentes geram leiautes diferentes.

As faixas funcionam como um repositório para os componentes que irão mostrar as informações. As faixas organizam o leiaute. Quando a faixa é impressa, os componentes colocados dentro dela são impressos também da mesma forma que foram organizados dentro da faixa. Em outras palavras, as faixas formam um macro-leiaute e os componentes dentro dela formam um micro-leiaute.

**OBS:** Tudo o que for colocado dentro da faixa será impresso exatamente da forma que foram colocados. Isso inclui fonte, posição e etc.

Na imagem do relatório acima, pode-se ver alguns componentes colocados dentro da faixa de cabeçalho, onde será impresso informações de nome de relatório, data, hora e logo da instituição.

#### **Componentes QuickReport:**

- Componente para relatório: Componentes básico para gerar novo relatório.

Ícone	Nome componente	Descrição
	QuickRep	Componente básico para o relatório. Sobre este componente é construído o relatório. O relatório padrão já tem esse componente. É

		associado a uma consulta pela propriedade Dataset.
	QuickAbstractReport	Não utilizado. Versão abstrata do componente QuickRep.
	QRCompositeReport	Não utilizado. Permite combinar múltiplos relatórios em um único documento.

**OBS:** O componente QuickReport deve ser associado a uma consulta para que possa mostrar os dados corretamente.

- Faixas: Componentes para organizar o layout do relatório.

Ícone	Nome componente	Descrição
	QRBand	Faixa básica e mais importante para criação de relatório. Sua propriedade "BandType" permite que seja alterada para qualquer tipo de faixa. O Tipo "rbDetail" trabalha com a consulta do relatório. Para cada registro da consulta associada ao relatório, a banda será repetida, mostrando as informações do registro atual. A consulta é a associada com o QuickRep pela propriedade Dataset.
	QRSubDetail	Faixa para criação de sub-detalhe de cada registro da consulta principal. É associada com uma consulta secundária cujos dados tem vínculo com os registros da consulta principal. Para cada registro da consulta principal, será impresso uma faixa QRSubDetail com as informações da consulta secundária. A consulta é associada à faixa pela propriedade Dataset. O link mestre-detalhe é feito entre as consultas.
	QRLoopBand	A faixa QRLoopBand age como um loop FOR. Quando a faixa é ativada, ela irá ser mostrada um número de vezes dependendo de seus parâmetros.  A propriedade "LoopIndex" age como a variável do loop e "LoopCount" especifica o número de impressões.
	QRStringBand	Permite carregar um texto em sua propriedade Items. Também implementa método LoadFromFile.
	QRChildBand	Esta faixa pode ser associada a outra faixa, criando uma faixa-filha, que pode ser controlada independentemente e que é impressa após a faixa associada. O jeito mais fácil de acrescentar um ChildBand é setar

		True na propriedade "HasShield" da faixa.
	QRGroupBand	Esta faixa tem a capacidade de criar "agrupamentos". Um agrupamento serve para dar mais visibilidade a uma informação que é igual para todo um grupo de registros. Cada vez que a informação muda, a faixa é impressa. A propriedade "Expression" determina qual a informação a ser monitorada (normalmente um campo da consulta). Esta faixa trabalha com uma faixa detalhe (QRBand tipo rbDetail) ou QRSubDetail. A propriedade "Master" determina qual faixa será

**OBS:** As faixas não contém informação por si só. A função das faixas é organizar o layout, arranjando as partes do relatório em si. Para mostrar uma informação (informação da consulta ou texto fixo), deve-se colocar componentes dentro das faixas que mostrem a informação.

- QRBand: O componente QRBand pode assumir diversos tipos de faixas diferentes dependendo de sua propriedade "BandType" listados abaixo:

BandType	Descrição
rbDetail	Principal tipo de faixa. Replicado para cada registro do DataSet associado à faixa.
rbTitle	Impresso uma vez no início do relatório.
rbPageHeader	Impresso uma vez no início de cada página.
rbPageFooter	Impresso no rodapé de cada página.
rbColumnHeader	Semelhante ao <i>rbPageHeader</i> exceto pelo fato que é impresso para cada coluna no caso de um relatório multicolumna.
rbSubDetail	Usado como <i>rbDetail</i> em relatórios mestre-detalhe.
rbGroupHeader	Impresso como cabeçalho de grupo de informações. Deve ser indicado em um <i>QRDetailLink</i> .
rbGroupFooter	Impresso como rodapé de grupo de informações. Deve ser indicado em um <i>QRDetailLink</i> .
rbSummary	Impresso no fim do relatório.

**OBS:** No lugar de *rbGroupHeader*, é mais conveniente utilizar o componente QRGroupBand. Este automaticamente se associa automaticamente ao frame designado como *rbDetail*.

**OBS:** No lugar de rbGroupFooter, é mais conveniente adicionar um QRBand e usar a propriedade "FooterBand" do QRGroupBand para transformar o QRBand em um GroupFooter automaticamente.

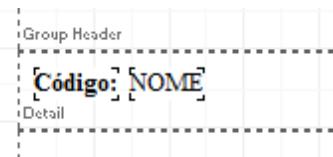
- Componentes para informações: São componentes que mostrar uma informação. Alguns exibem informações da consulta do relatório, outros informações do sistema (data e hora, número da página) e alguns tem funções específicas como mostrar um somatório ou expressão especial.

Ícone	Nome componente	Descrição
	QRLabel	Componente que mostra um texto estático. A propriedade "Caption" determina o valor do texto. A propriedade "Font" determina as características da fonte.
	QRDBText	Componente para mostrar o valor de uma coluna da consulta associada com a banda. Ajuste a propriedade "Dataset" para a consulta de origem dos dados e "DataField" para a coluna pretendida.
	QRExpr	Componente para gerar um informação dependendo de condições. A propriedade "Expression" permite criar comandos condicionais complexos, podendo por exemplo, testar valores específicos vindo da consulta e imprimir informações diferentes para cada caso.
	QRSysData	Este componentes tem a função de mostrar dados do sistema como a data/ hora do sistema ou página atual.
	QRMemo	Este componente permite visualizar textos grandes. Preencha a propriedade Lines para adicionar o texto.
	QRExprMemo	Componente que mistura o QRExpr e QRMemo. Permite a criação de expressões na propriedade Lines.
	QRRichText	Componente similar ao QRLabel, mas permite visualizar texto formatado (RichText).
	QRDBRichText	Componente que mistura o QRDBText com QRRichText. Associado a um Dataset e um DataField, permite visualizar uma informação RichText da consulta.
	QRShape	Permite desenhar formas no relatório. As formas podem ser quadrados, círculos, linhas...etc. A propriedade "Shape" determina a forma pretendida. "Pen" e "Brush" para definir as propriedades de cor, espessura da linha e etc. Muito usado para fixar linhas e caixas para dar mais ênfase em certas partes do relatório.
	QRImage	Permite mostrar uma imagem no relatório. A propriedade "Picture"

		armazena a imagem a ser mostrada.
	QRDBImage	Versão do QRImage para mostrar uma imagem do banco de dados. Assim como o QRDBText, basta associar a consulta ao "Dataset" e a coluna em "DataField".

### Adicionando componentes em uma faixa de relatório:

Ao adicionar um componente de informação em uma faixa, deve-se definir os detalhes desse componente dentro da faixa. Dependendo do componente, deve-se ainda associar à coluna da consulta ou definir o valor a ser mostrado. Veja exemplo abaixo:



Componente QRLabel e QRDBText

No exemplo, foi adicionado à faixa definida como detalhe, um QRLabel e um QRDBText. Para o QRLabel, foi alterado a posição (prop. "Top" e "Left") e fonte (prop. Font). Para o QRDBText, foi alterado o fonte e associado o componente à uma coluna da consulta através das propriedades Dataset e Datafield.

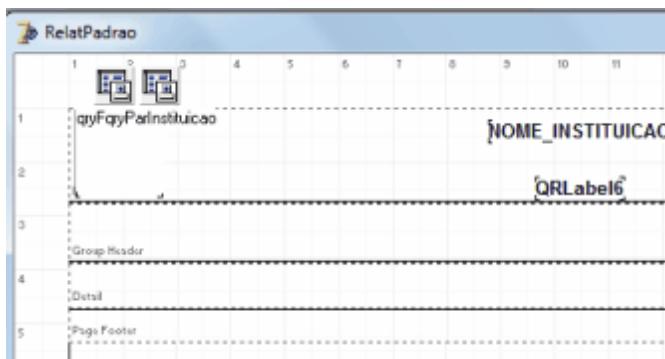
Existem diversas propriedades para os componentes de informação e faixas. As propriedades específicas dos componentes para QuickReport são listadas abaixo:

Propriedade	Descrição
Dataset	Propriedade que informa a consulta utilizada pelo componente para procurar a informação a ser exibida. Existe em todos os componentes QRDB*. Existente também no componente QuickRep e em alguns tipos de faixas como QRSubDetail.
Datafield	Indica a coluna da consulta que o componente deve procurar a informação. Existe em todos os componentes QRDB*.
Expression	Permite criar uma expressão complexa para mostrar uma informação. Essa expressão contém testes, cálculos e acesso a diversas funções. Pode trabalhar com informações da consulta associada ao QuickRep.
Master	Aparece em algumas faixas como QRGroupBand. Indica qual a faixa tipo detalhe que a faixa de agrupamento deve ser associada.
FooterBand	Propriedade da faixa QRGroupBand. Serve para associar um QRBand como rodapé do agrupamento.
LinkBand	Propriedade de faixa para associar outra faixa. Indica que a faixa deve

	ser impressa na mesma página que a faixa associada. Se não houver espaço para ambas as faixas, então uma nova página é criada.
HasChild	Propriedade de faixa. Indica se esta deve ter uma faixa filha. Ao passar para True, é criado uma faixa automaticamente. Esta é impressa logo após a faixa pai. Isso permite ajustar configurações para um espaço do layout que difere da faixa pai, mas que vai aparecer sempre junto.
ForceNewPage	Propriedade de faixa. Força com que seja criado uma nova página antes da impressão da faixa. Útil por exemplo em faixas de agrupamento.
ResetAfterPrint	Aparece em QRExpr. Se verdadeiro o valor de uma expressão somatório será resetado após ser exibido. Útil para mostrar totais em rodapé de agrupamentos.
Picture	Aparece em QRImage. Abre uma caixa de diálogo para localizar a imagem a ser exibida.
Lines	Aparece em QRMemo e QRRichText. Abre uma caixa de diálogo para escrever o texto a ser exibido.
Shape	Aparece em QRShape. Indica o formato da forma a ser desenhada.

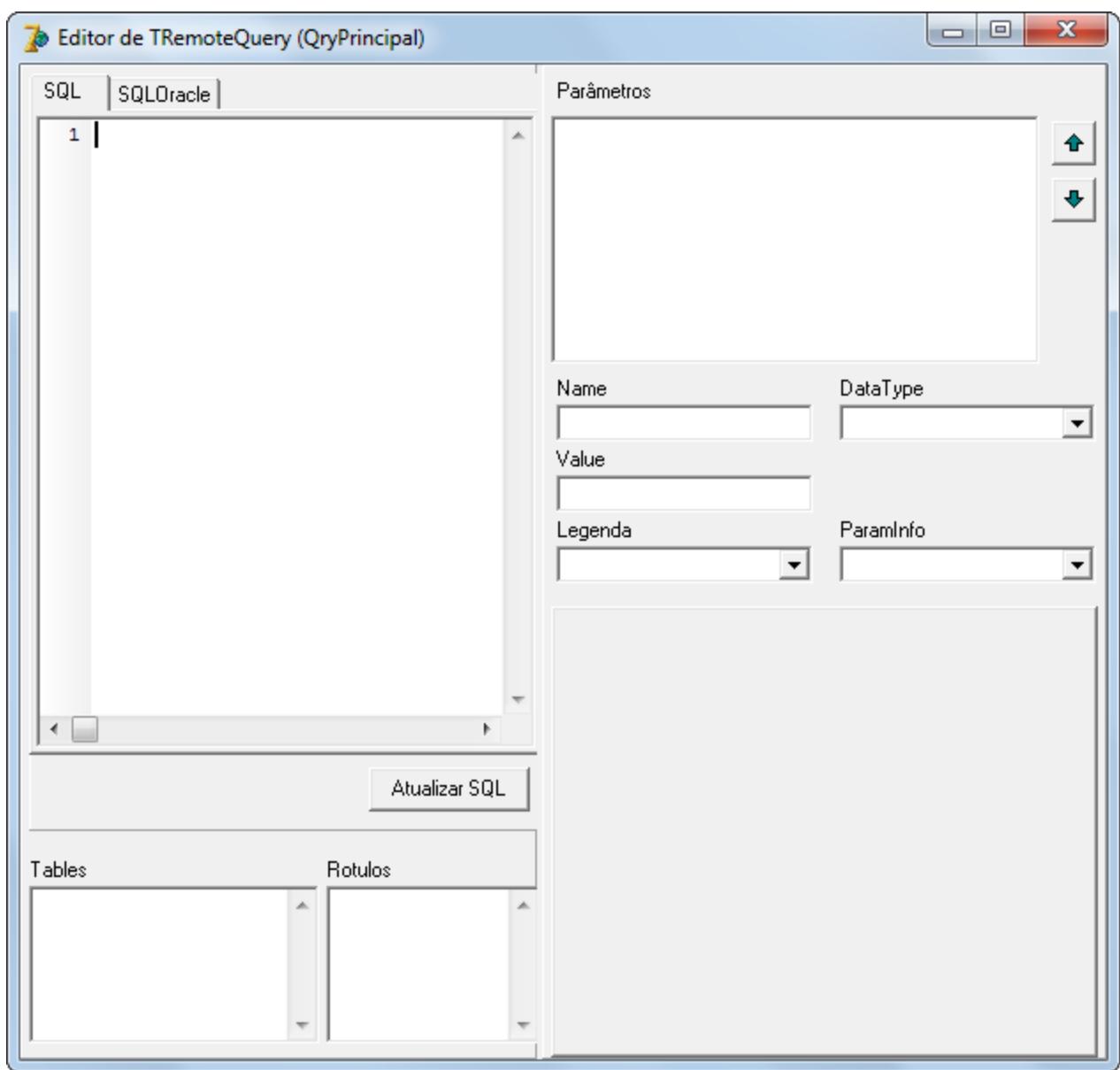
### 5.1.2 Consulta do relatório

Um relatório QuickReport é por definição associado a um dataset que trará os dados do banco de dados para exibição. O layout padrão já traz dois datasets para uso, e mais pode ser adicionado se necessário.



Consultas padrão

O dataset usado é um TRemoteQueryRep, que herda de TQuery. Foi adicionado algumas características específicas para facilitar o desenvolvimento do relatório. Ao clicar duas vezes sobre um dos componentes, a seguinte janela será mostrada:



Tela de configuração da consulta

Nessa janela, podemos inserir a consulta (texto SQL), definir as propriedades dos parâmetros e ainda se é utilizado restrições na mesma.

Os campos da tela são:

**SQL e SQLORACLE:** Caixa para incluir a consulta SQL. Após, clique em "Atualizar SQL" para os parâmetros sejam atualizados na lista da direita. o SQLORACLE deve ser preenchido apenas se a consulta deve ser diferenciada para este banco.

**Tables e Rótulos:** Permitem adicionar restrições à consulta.

**Parâmetros:** Lista de parâmetros da consulta. Estes parâmetros são mostrados ao usuário

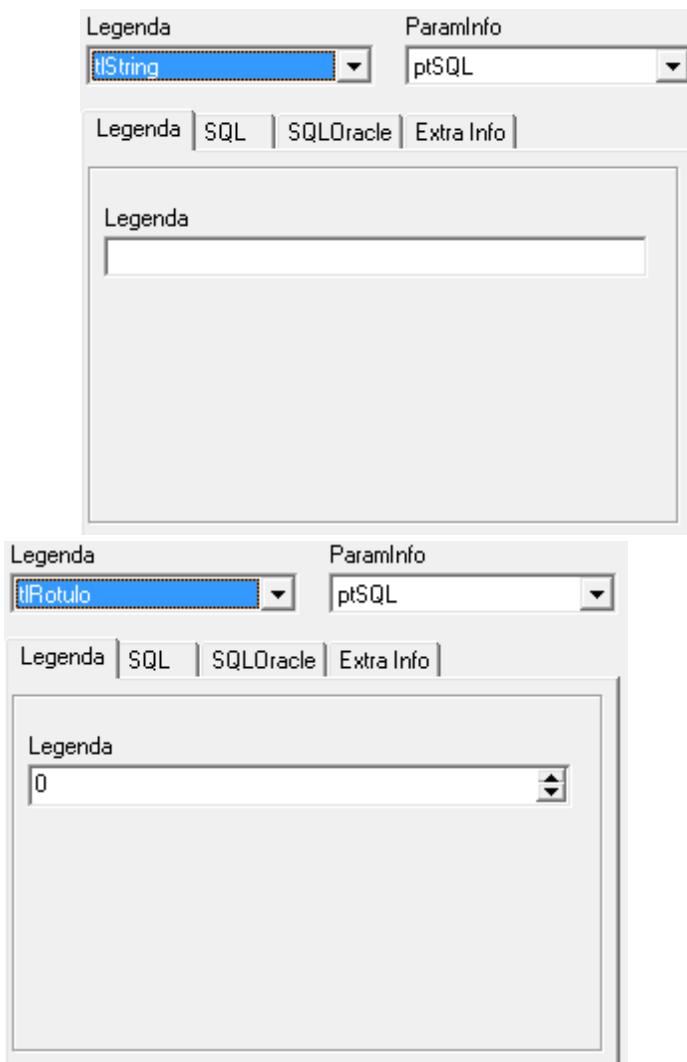
ao executar o relatório.

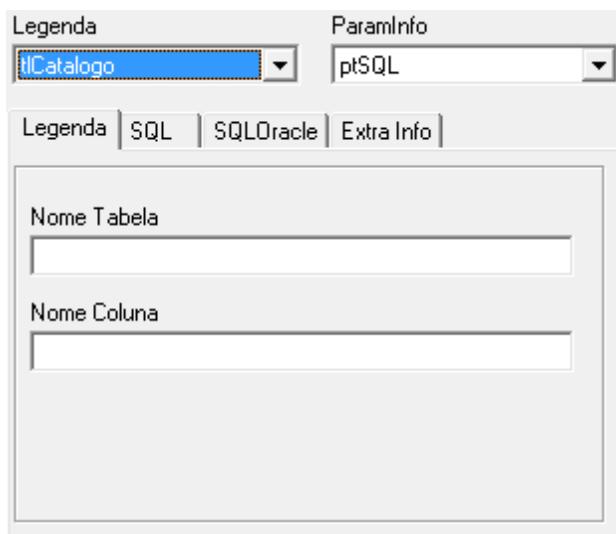
Para cada parâmetro é definido:

**Name:** Nome do parâmetro. Não é necessário alterar.

**DataType:** Tipo de informação do parâmetro. Entre os diversos tipos, os mais comuns são ftInteger, ftDate, ftString. Dependendo do tipo informado, o parâmetro será mostrado ao usuário em um componente específico. Se for ftDate por exemplo, será mostrado um calendário para escolher a data.

**Legenda:** Legenda do parâmetro mostrado na tela de parâmetros do relatório. Define se o texto é fixo (tlString), se vem do catálogo (tlCatalogo) ou de rótulo cadastrado no sistema (tlRotulo).





Opções de legenda para o parâmetro. Texto, Rótulo ou Catálogo.

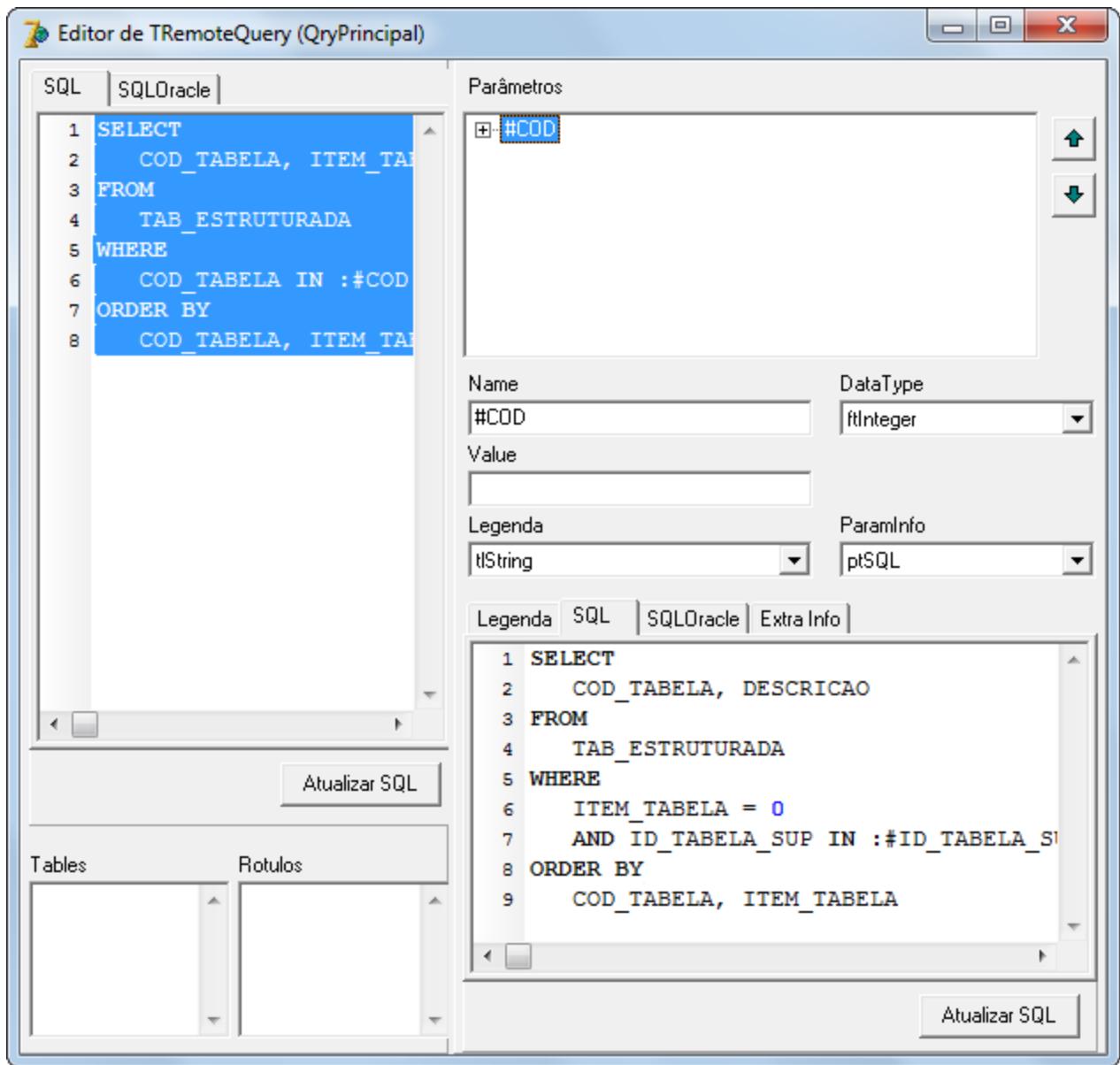
**ParamInfo:** Define se é apresentado ao usuário opções para escolher ou se o usuário insere o valor do parâmetro manualmente. Pode ser ptNormal (entrada manual) ou ptSQL (lista de valores para o usuário escolher).

Se for escolhido ptSQL, será mostrado outras abas (SQL e SQLOracle) junto à aba de legenda. Nela, é possível adicionar um SQL para listar opções para o usuário escolher. Se o parâmetro da consulta for de múltipla escolha (COLUNA IN #PARAM), então o usuário poderá escolher vários entre as opções listadas.

**OBS:** As consultas do parâmetro podem ter parâmetros também, levando a uma "árvore" de dependência entre os parâmetros. Após preencher ou selecionar um valor no primeiro, o seguinte será habilitado ou preenchido para escolha.

**OBS:** Somente se todos os parâmetros forem preenchidos é que o relatório poderá ser executado.

A imagem abaixo mostra um exemplo de parâmetros com dependência entre parâmetros:



#### Parâmetros com dependência

**Extra info:** Permite adicionar restrição à consulta do parâmetro. Também permite configurar a listagem dos valores do parâmetro tipo SQL em uma árvore se a tabela da consulta assim permitir.

**OBS:** Para que a listagem do parâmetro seja mostrado em árvore, a tabela deve ter uma coluna que se refere à coluna primária da tabela, de forma que certos registros teriam uma relação pai-filho. Por exemplo, a TAB\_ESTRUTURADA tem as colunas ID\_TABELA (primária) e ID\_TABELA\_SUP.

Na consulta do parâmetro, a primeira coluna deve ser ID\_TABELA, e a última ID\_TABELA\_SUP. A segunda coluna (DESCRICAO) será mostrada ao usuário. Na configuração

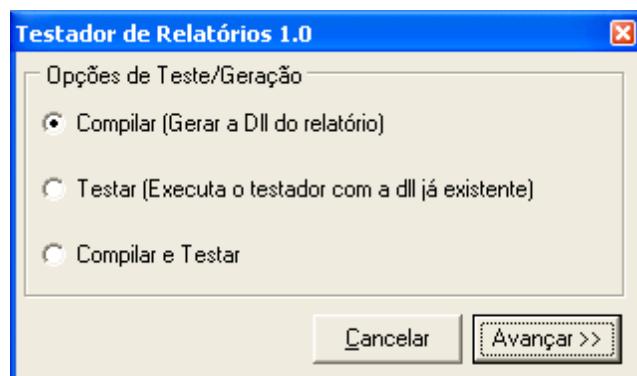
da aba "Extra info", selecione ID\_TABELA\_SUP.

### 5.1.3 Testando o relatório

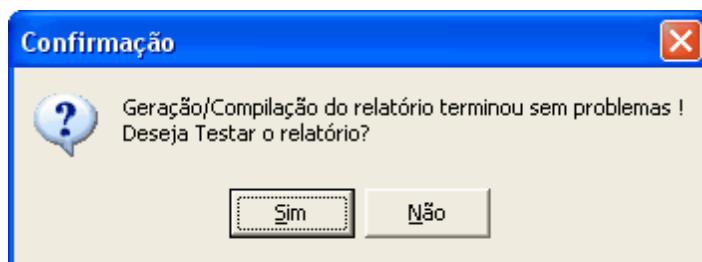
Para compilar e testar relatórios do SI\*, devemos utilizar o menu "CDESP Experts" - "Report Builder" - "Local Builder":



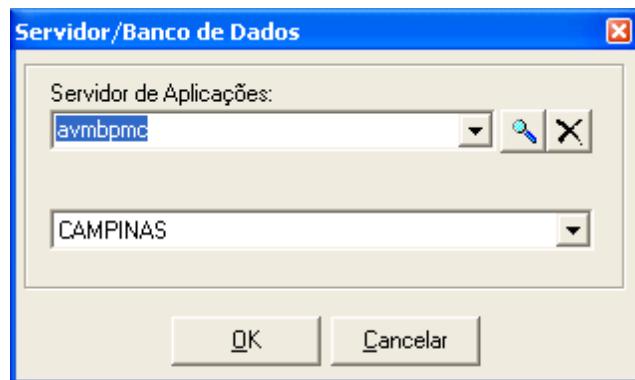
Uma nova tela se abrirá:



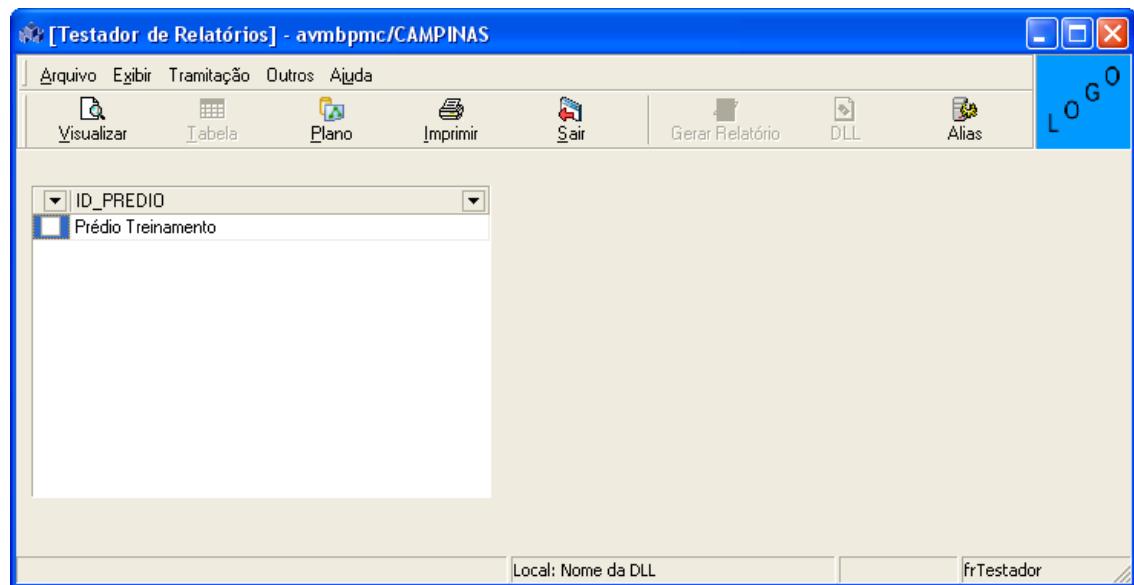
Escolha a opção "Compilar e Testar". Se não houver erros durante a compilação, a seguinte tela irá aparecer:



Na próxima tela, selecione o servidor de aplicações e o alias do banco que será utilizado para testes:



Na próxima tela serão mostrados todos os parâmetros do relatório:



Selecione os parâmetros e clique em Visualizar:



## 5.2 Relatórios RTF

Relatórios RTF foram criados para atender a necessidade de poder gerar documentos formatados para impressão a partir do SI\*, cuja origem das informações seja unicamente do próprio banco de dados. Dessa forma, pode-se ter documentos oficiais como diplomas e certificados, padronizados, diminuindo assim a ocorrência de erros.

Por não serem compilados, relatórios RTF podem ser criados pelo próprio cliente, agilizando o desenvolvimento e facilitando a personalização. Ainda, podendo ser desenvolvido em editor de texto nativo do Windows®, não é necessário a instalação de aplicações de terceiros, diminuindo assim os custos para a sua criação.

### 5.2.1 Introdução

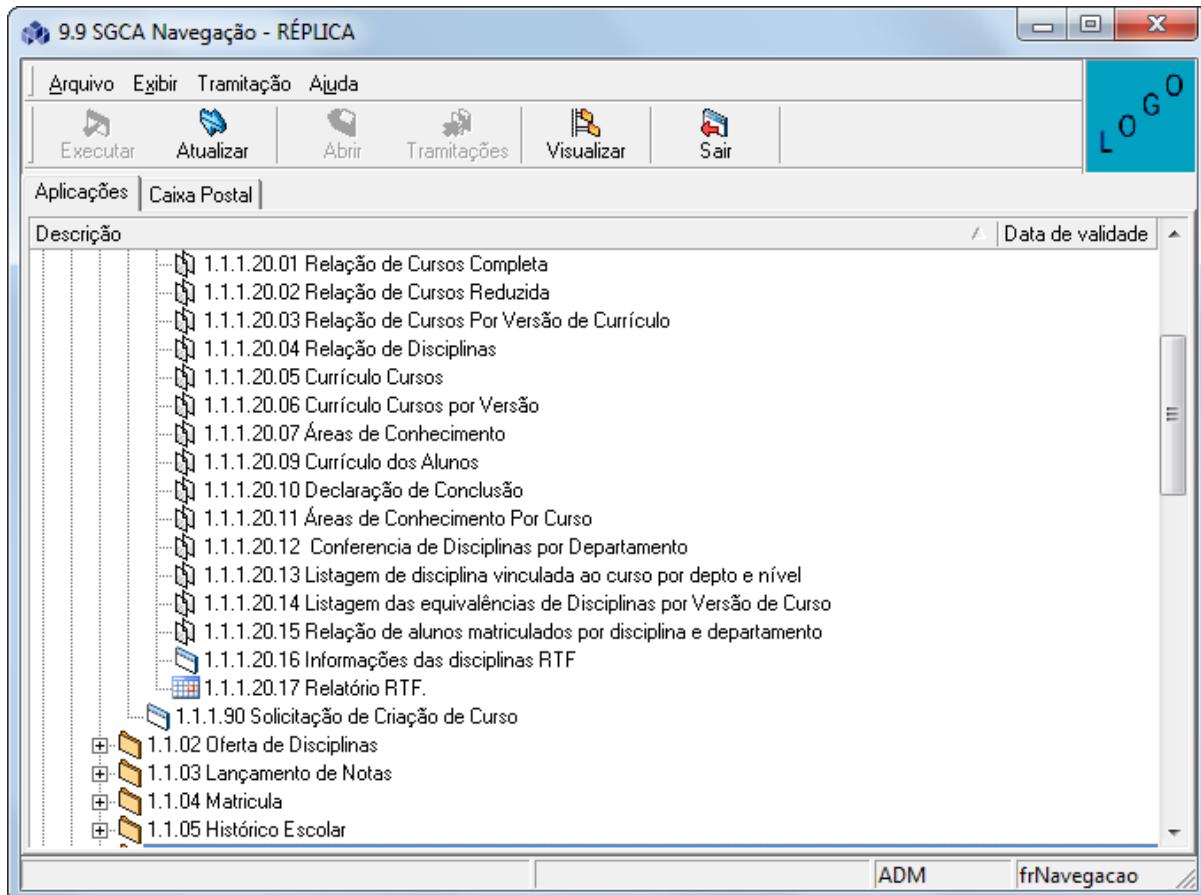
Imagine o seguinte processo: Um funcionário chega ao departamento de RH e pede um certificado de participação de uma atividade da empresa. O funcionário abre um documento padrão contido em seu computador. Requisita os dados do funcionário e preenche naquele momento as informações do certificado. Por fim, manda para a impressão.

Agora, adicionemos a necessidade de realizar isso para todos os funcionários do departamento. Mesmo para vinte ou trinta, o trabalho acaba sendo longo e sujeito a muitos erros. E se for necessário alterar o leiaute do documento e outros funcionários tiverem suas próprias cópias? Deve-se então atualizar todos os documentos espalhados em vários computadores.

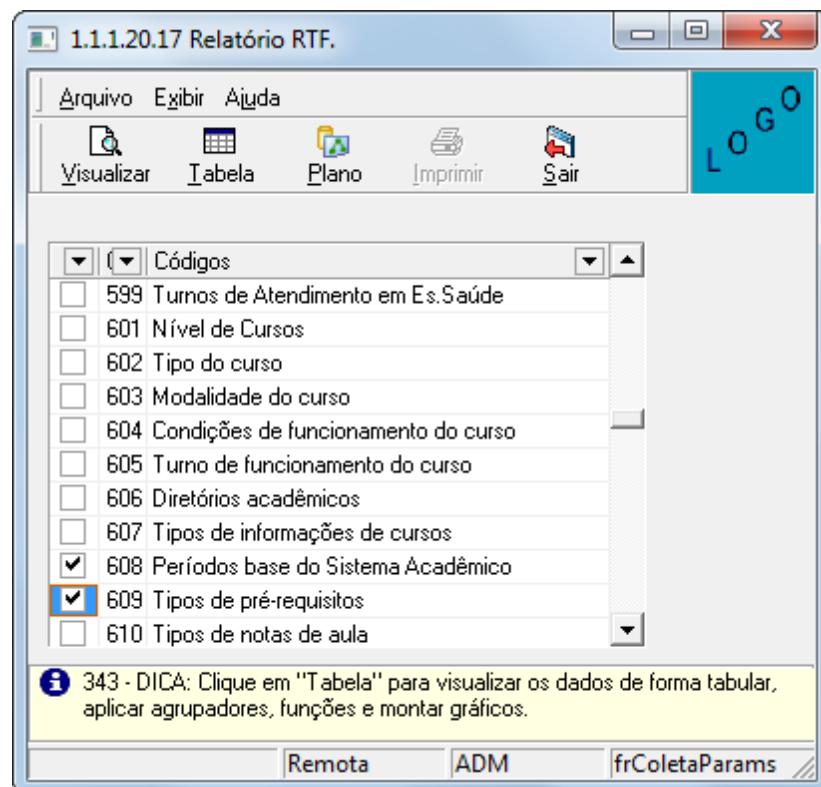
Um relatório RTF do SI\* pode suprir essas necessidades de gerenciamento e organizar os documentos a serem impressos. O relatório fica guardado no próprio sistema, centralizando os documentos usados. Sendo gerado automaticamente, tem as informações com origem no banco de

dados do sistema, diminuindo a possibilidade de erros de digitação. E se necessário, basta alterar o leiaute e todos os relatórios passarão a ser gerados com as alterações feitas.

Abaixo um exemplo de relatório sendo executado no sistema:



Relatório RTF sendo exibido na árvore do SI\*.



Tela de parâmetros do relatório.

The screenshot shows a Microsoft Word document window titled "Rel5DF7.tmp.rtf : 2 - BrOffice.org Writer". The menu bar includes "Arquivo", "Editar", "Exibir", "Inserir", "Formatar", "Tabela", "Ferramentas", "Janela", and "Ajuda". The toolbar includes icons for file operations, magnifying glass, and other functions. The status bar shows "77%" and "Fechar visualização".

The document content includes:

- A logo in the top left corner.
- A header section with the text "Universidade Federal de Santa Maria" and "1.1.1.20.17 Relatório RTF."
- A timestamp "Data : 24/09/2010" and "Hora : 11:20".
- A large table with columns "Cod.", "Item", and "Descrição". The table lists various codes and descriptions, such as:
  - 3 0 Retornar em outra ocasião
  - 5 0 Holanda
  - 608 0 Períodos base do Sistema Acadêmico
  - 608 100 Anual
  - 608 101 1. Semestre
  - 608 102 2. Semestre
  - 608 103 Curso de Verão
  - 608 104 Curso de Inverno
  - 608 200 Semestral
  - 608 300 Trimestral
  - 608 301 1. Trimestre
  - 608 302 2. Trimestre
  - 608 303 3. Trimestre
  - 608 304 4. Trimestre
  - 608 400 Etapas
  - 608 401 Etapa 1
  - 608 402 Etapa 2
  - 608 403 Etapa 3
  - 608 404 Etapa 4
  - 609 0 Tipos de pré-requisitos
  - 609 1 Disciplina
  - 609 2 Co-matriz

Relatório RTF em execução

### 5.2.2 Componentes

O relatório RTF consiste em um leiaute, criado em um editor de texto, salvo em formato ".rtf" e uma consulta SQL. Esses dois itens são cadastrados no SI\* pelo cadastro de aplicações e o resultado é um item na árvore do SGCA que executa o relatório, da mesma forma que um relatório padrão.

Os detalhes do cadastro são mostrados abaixo:

Primeiro, se cadastra o item relatório, usando o cadastro de aplicações:

The screenshot shows the '01.02.03 Cadastro de Aplicações' window. The menu bar includes 'Arquivo', 'Exibir', 'Tramitação', 'Outros', and 'Ajuda'. The toolbar has icons for 'Novo', 'Alterar', 'Excluir', 'Localizar', 'Propriedades', 'Novo', 'Visualizar', and 'Copiar'. A logo is visible in the top right corner. The main area is titled 'Aplicação' and contains tabs for 'Aplicação', 'Restrições de Dados', 'Restrições de Funcionalidade', 'Relatórios da Aplicação', and 'Config. Componente'. The 'Aplicação' tab is selected. It has fields for 'Descrição' (Description), 'Fonte' (Source) set to 'Aplicação superior' (Superior Application), and 'Tipo' (Type). The 'Tipo' section includes radio buttons for 'Menu', 'Relatório', 'Internet', 'Caixa Postal', 'Externa', 'Principal', 'Formulário', 'Ação', 'Perfil (uso interno)', and 'Parâmetro de Relatório'. Below these are sections for 'Aplicação de Abertura (Relatórios)', 'Cabeçalho', 'Consulta Relatório', and 'Arquivo'. There are also fields for 'Validade da Mensagem' (Message Validity), 'Auditada' (Audited), 'Situação' (Status), and 'Mensagem' (Message) with a rich text editor. An 'Ícone' (Icon) section shows a placeholder icon. At the bottom are 'Salvar' (Save) and 'Cancelar' (Cancel) buttons, and status bars for 'APLICACOES - MENSAGEM', 'ADM', and 'frSGCAMAplica'.

#### Cadastro de Aplicações

O cadastro de aplicações é usado para criar um item na árvore do SI\*, de forma que poderá ser executado e exibir o relatório. Os campos que devem ser preenchidos são:

**Descrição:** Nome do relatório na árvore do SI\*.

**Aplicação Superior:** Posição do relatório na árvore do SI\*.

**Tipo:** Habilita os campos correspondentes na aplicação. Deve ser "Relatório".

**Consulta Relatório:** Realiza o cadastro da consulta do relatório. Explicado mais abaixo.

**Arquivo:** Realiza o cadastro do arquivo de leiaute. Explicado mais abaixo.

Deve-se preencher os itens descrição, aplicação superior e tipo como na imagem abaixo:

**9.13 Cadastro de Aplicações**

Arquivo Exibir Tramitação Outros Ajuda

Novo Alterar Excluir Localizar Propriedades Novo Visualizar Copiar

Aplicação: 1.1.1.20.17 Relatório RTF.

Aplicação | Restrições de Dados | Restrições de Funcionalidade | Relatórios da Aplicação | Config. Componente |

Descrição  
1.1.1.20.17 Relatório RTF.

Fonte Aplicação superior  
1.1.1.20 Relatórios

Tipo

<input type="radio"/> Menu	<input checked="" type="radio"/> Relatório	<input type="radio"/> Web	<input type="radio"/> Caixa Postal	<input type="radio"/> Externa
<input type="radio"/> Principal	<input type="radio"/> Template	<input type="radio"/> Ação	<input type="radio"/> Perfil	<input type="radio"/> Parâmetro

Aplicação de Abertura (Relatórios) Cabeçalho Consulta Relatorio Arquivo

Validade da Mensagem Auditada Situação  
23/09/2010 Sim Ativa

Mensagem

Salvar

004 - 004 - Alterando ADM frS

Preenchendo os campos do relatório

O próximo item a definir é a consulta do relatório, usando o campo "Consulta Relatório" conforme abaixo:



Campo consulta do relatório

As funções dos botões são:

**Localizar:** Localiza uma consulta já cadastrada.

**Novo:** Cadastra nova consulta no sistema. A mesma consulta pode ser usada em mais de um relatório.

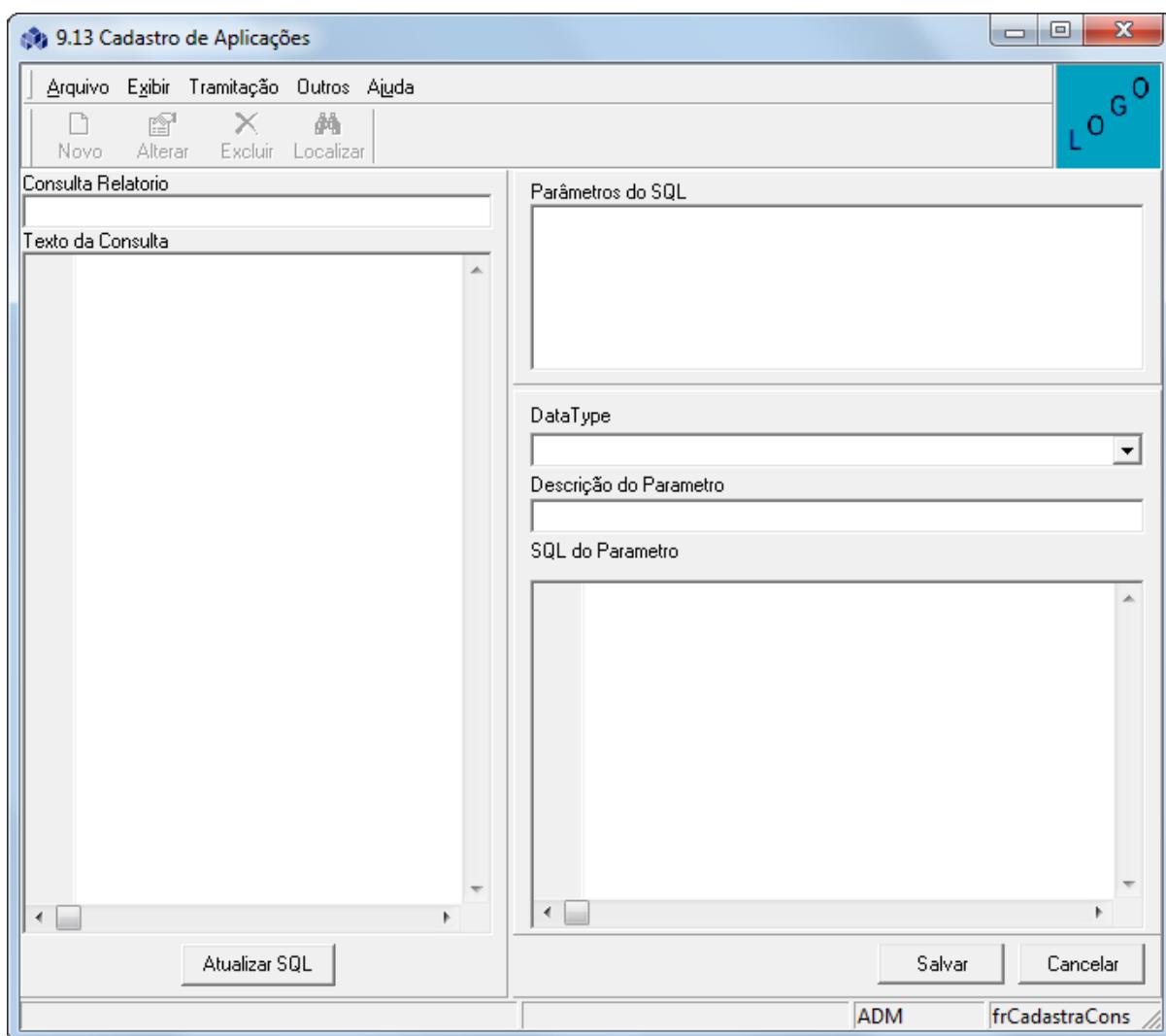
**Propriedades:** Abre o cadastro de consulta para visualizar os detalhes da consulta atual.

Se clicar em localizar, abre-se a seguinte janela:



Exemplo localizar consulta

Clicando no botão Novo temos a seguinte janela:



#### Cadastro de consulta de relatório

Neste cadastro é definido a consulta, assim como consultas usadas em parâmetros para a consulta principal. Os campos dessa janela são:

**Consulta Relatório:** Nome da consulta. Usada ao localizar a mesma pelo cadastro de aplicações.

**Texto da Consulta:** Consulta SQL do relatório. Só pode haver uma por relatório.

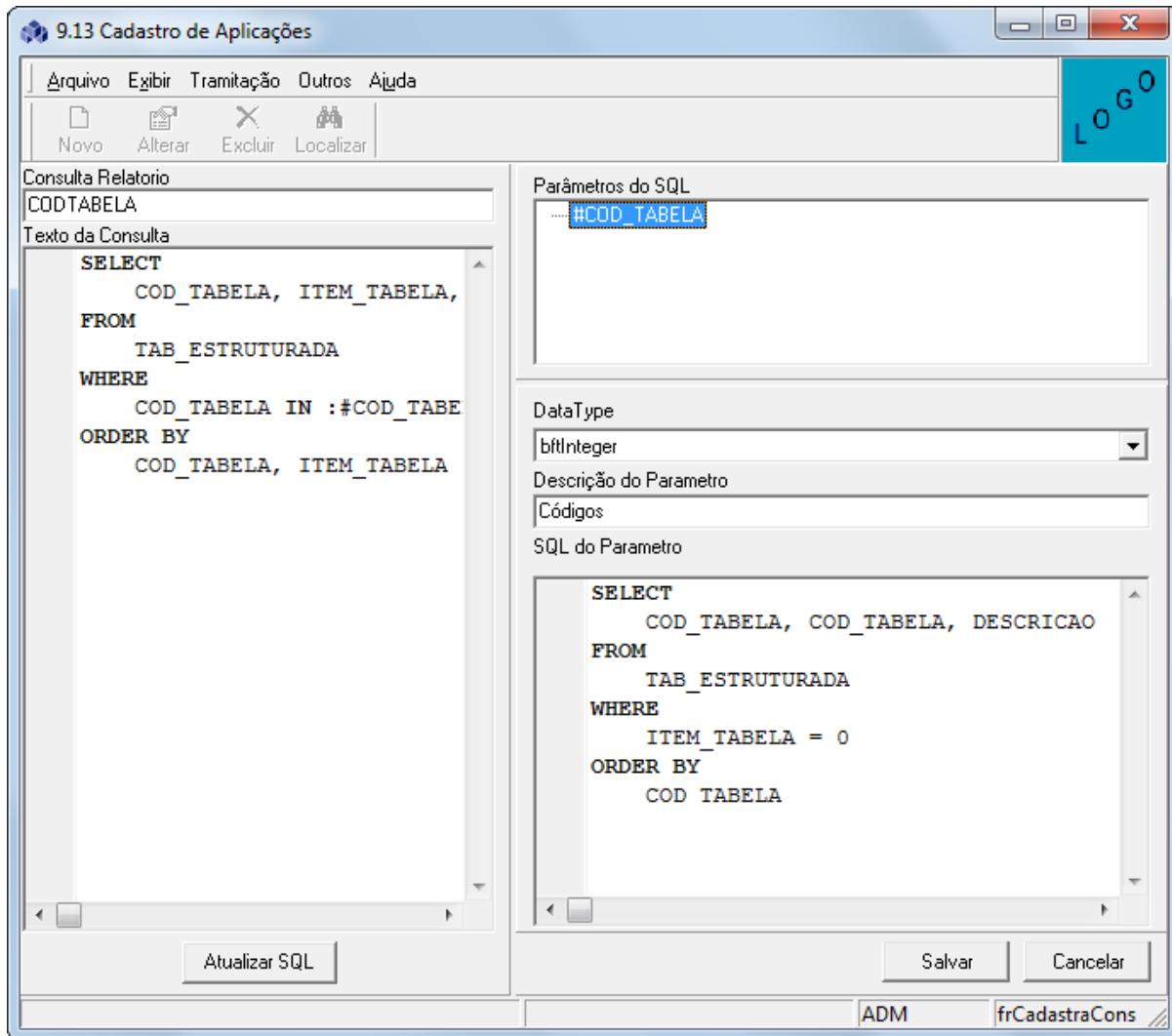
**Parâmetros do SQL:** Se a consulta tiver parâmetros, eles serão identificados nesta área, onde poderão ter suas características definidas nos campos abaixo. Os parâmetros aparecem para o usuário escolher ao executar o relatório.

**Data Type:** Define o tipo de parâmetro. Pode ser do tipo texto, numérico, data e etc. O tipo do parâmetro define o tipo de opção que será apresentado ao usuário. Pode ser um calendário, valor numérico ou listagem de itens.

**Descrição do Parâmetro:** Nome mostrado ao usuário na tela de parâmetros.

**SQL do Parâmetro:** Texto da consulta para o parâmetro. Se não houver texto, será apresentado ao usuário uma caixa para inserção do valor.

Abaixo um exemplo de consulta preenchida:



Janela de parâmetros preenchido

Clique no botão Salvar e feche a janela.

Voltando ao cadastro de aplicações, deve-se ainda cadastrar o arquivo de leiaute, usando o campo Arquivo abaixo:



Campo Arquivo

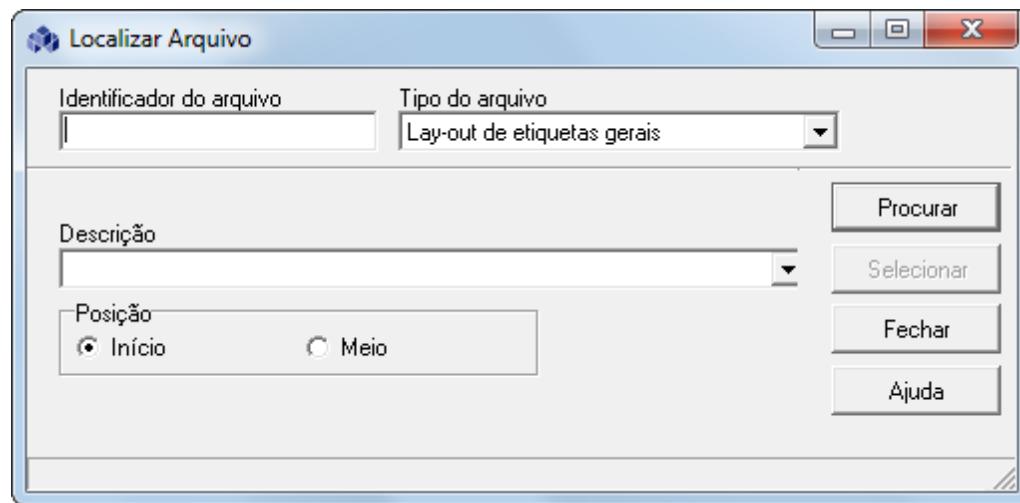
Novamente, temos os botões localizar, novo e propriedades cujas funções são:

**Localizar:** Localiza um arquivo de leiaute já cadastrada.

**Novo:** Cadastra novo leiaute no sistema. O mesmo arquivo pode ser usada em mais de um relatório.

**Propriedades:** Abre o cadastro de arquivos para visualizar os detalhes do arquivo atual.

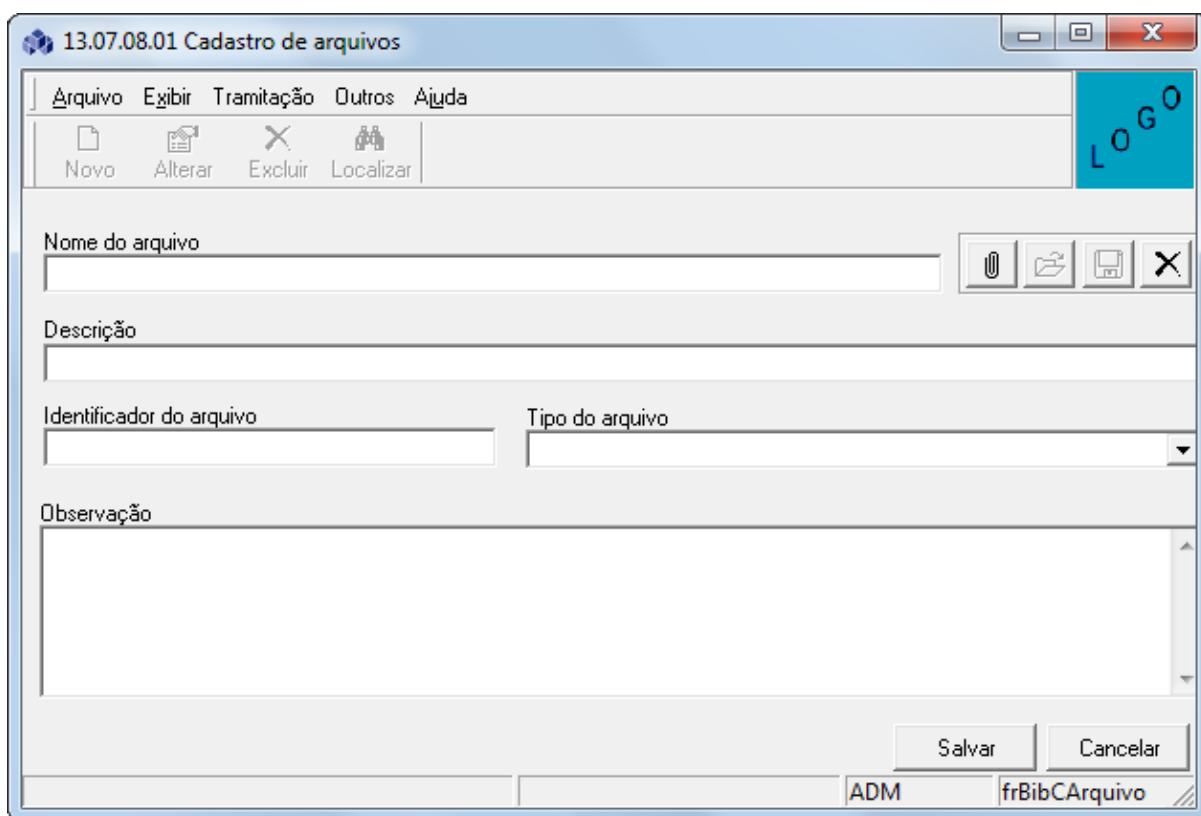
Se clicar no botão localizar temos a seguinte janela:



Exemplo localizar arquivo

Pode-se localizar por descrição ou identificador do arquivo.

Clicando no botão Novo temos a seguinte janela:



Janela de cadastro de arquivos

Nesta janela temos os seguintes campos:

**Nome do arquivo:** Identifica o arquivo fisicamente. É preenchido automaticamente ao localizar o arquivo no computador.

**Descrição do arquivo:** Campo para preencher uma descrição.

**Identificador do arquivo:** Usado no localizar do arquivo. Usualmente pode-se optar por localizar pela descrição, nome ou identificador.

**Tipo de arquivo:** Para relatórios RTF deve-se selecionar "Lay-Out de relatórios".

**Observação:** Campo para preencher uma descrição maior.

Os botões ao lado do campo **Nome do arquivo** tem as seguintes funcionalidades:

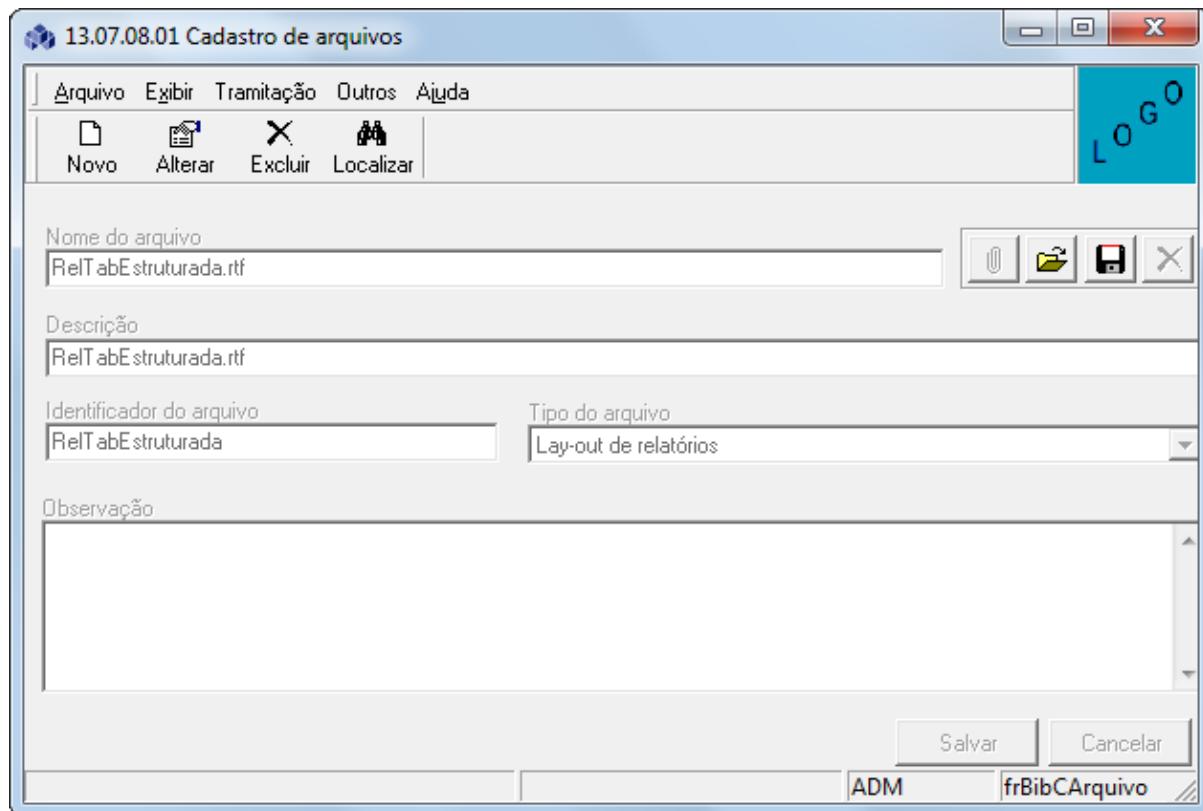
**Clipe:** Localizar e associa o arquivo de leiaute ao cadastro. O arquivo não pode estar sendo editado por outro programa ao ser localizado.

**Pasta:** Abre o arquivo cadastrado. Caso se deseje editar o mesmo, deve-se salvar em uma pasta usando o botão do disquete. O arquivo salvo pode então ser alterado. Por fim, usar o botão de clipe para o mesmo ser associado novamente ao cadastro.

**Disquete:** Salva o arquivo cadastrado no computador.

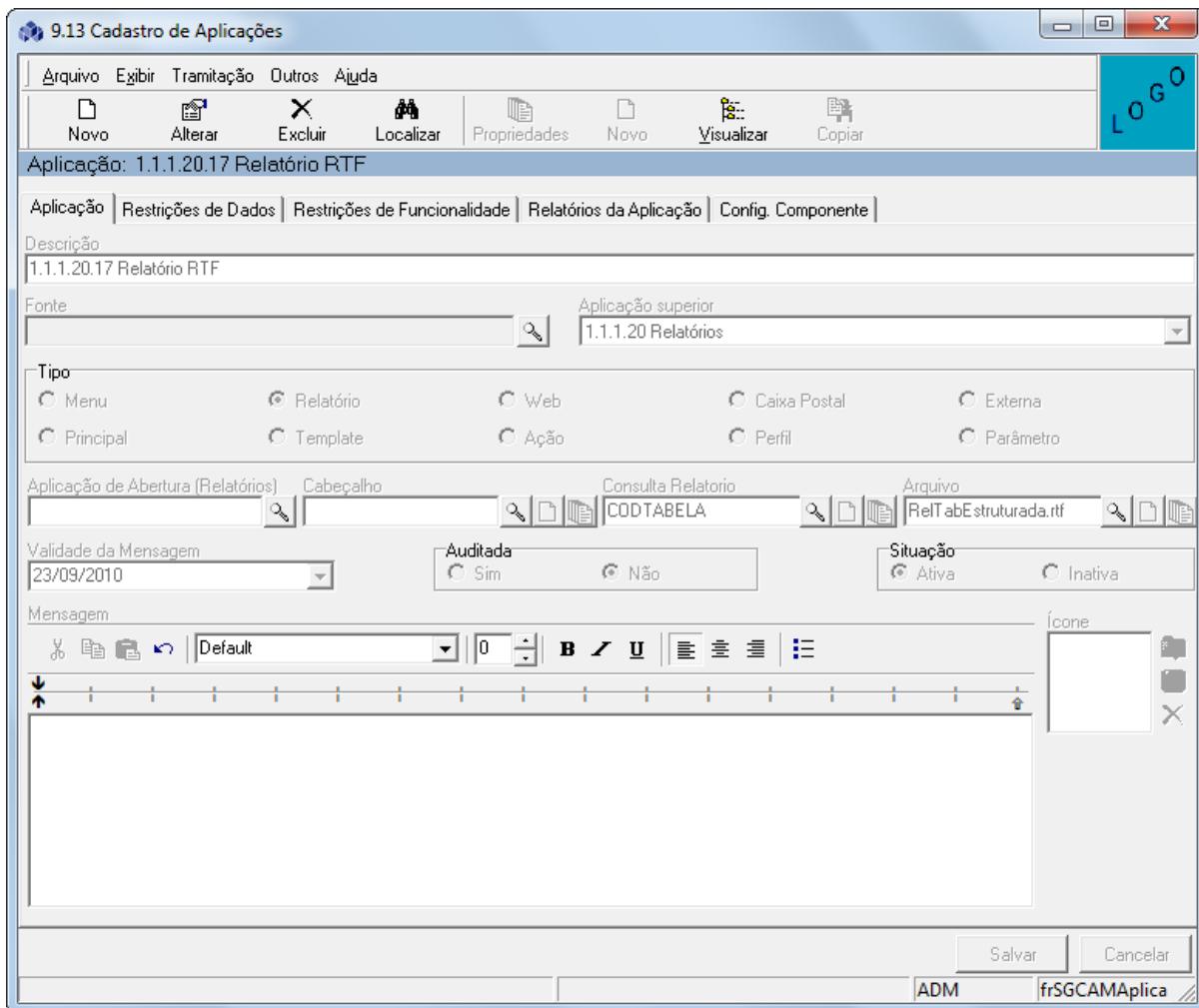
X: Limpa a informação de leiaute cadastrada.

Abaixo um exemplo de cadastro realizado:



Cadastro de arquivo preenchido

Após salvar, feche a janela e retorne para o cadastro de aplicações e clique em salvar. A janela estará como a seguir:



#### Finalizando o cadastro do relatório RTF

Após atualizar o SGCA o relatório deverá aparecer no local especificado da árvore do SI\*.

### 5.2.3 Consulta

O relatório RTF aceita uma consulta SQL, com ou sem VIEWS, que retorne uma ou mais colunas. Não é permitido utilizar UPDATEs, DELETEs ou outro tipo de SCRIPT para o relatório.

A consulta é cadastrada através do cadastro consultas acessado pelo cadastro de aplicações de pode ou não ter parâmetros, conforme explicado no tópico "Componentes".

Para o exemplo seguinte foi cadastrado a seguinte consulta:

**SELECT**

COD\_TABELA AS COL\_A, ITEM\_TABELA AS COL\_B, DESCRICAO AS COL\_C

**FROM**

TAB\_ESTRUTURADA

**WHERE**

COD\_TABELA IN #COD\_TABELA

**ORDER BY**

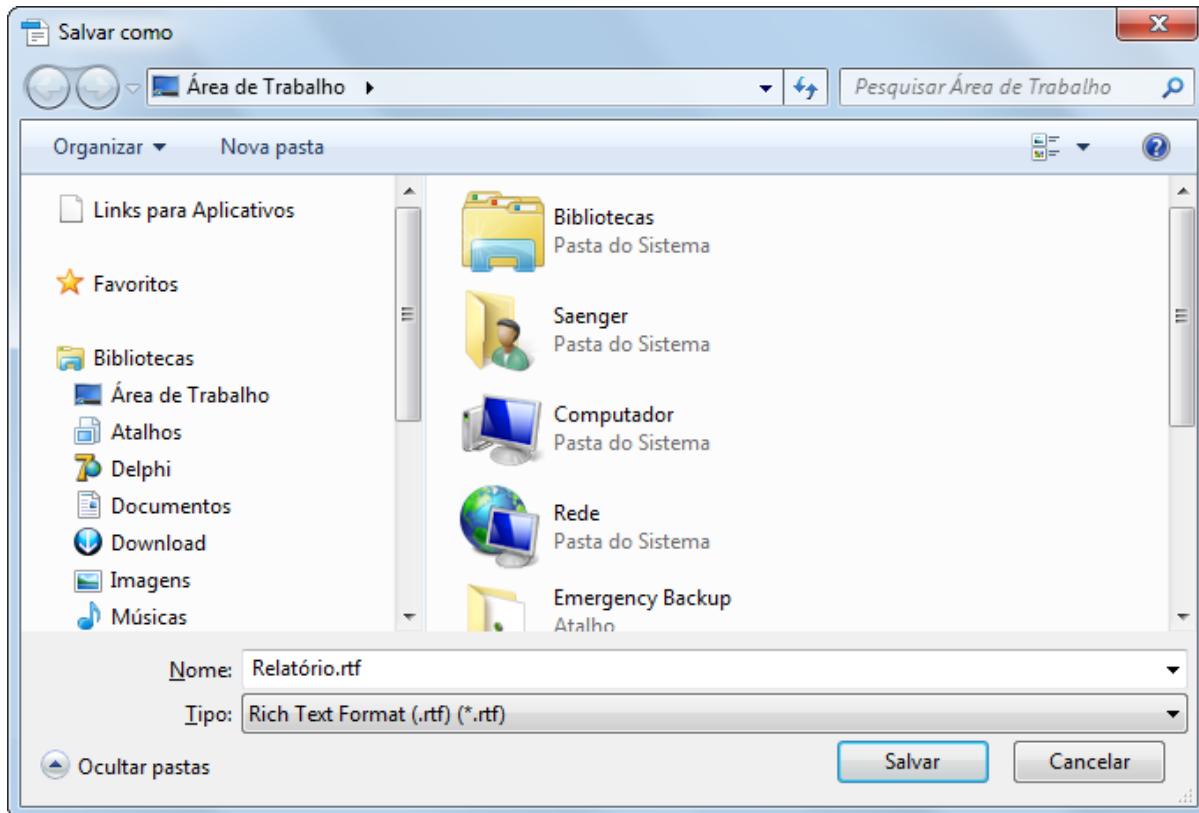
COD\_TABELA, ITEM\_TABELA

**5.2.4 Leiaute**

O leiaute do relatório RTF é um arquivo ".rtf" que pode ser criado em diversos editores de texto. Dessa forma, o relatório RTF permite que se obtenha qualquer efeito permitido pelo editor, possibilitando obter relatórios com formatação muito superior aos relatórios padrão, de forma a gerar diplomas, certificados e outros documentos que necessitam de visual mais trabalhado.

Para criar um leiaute, abra um editor de texto e salve o arquivo no formato "Rich Text Format"(.rtf). Como exemplo, é usado o OpenOffice.

**OBS:** Somente arquivos salvos como "Rich Text Format" serão aceitos para a execução do relatório.

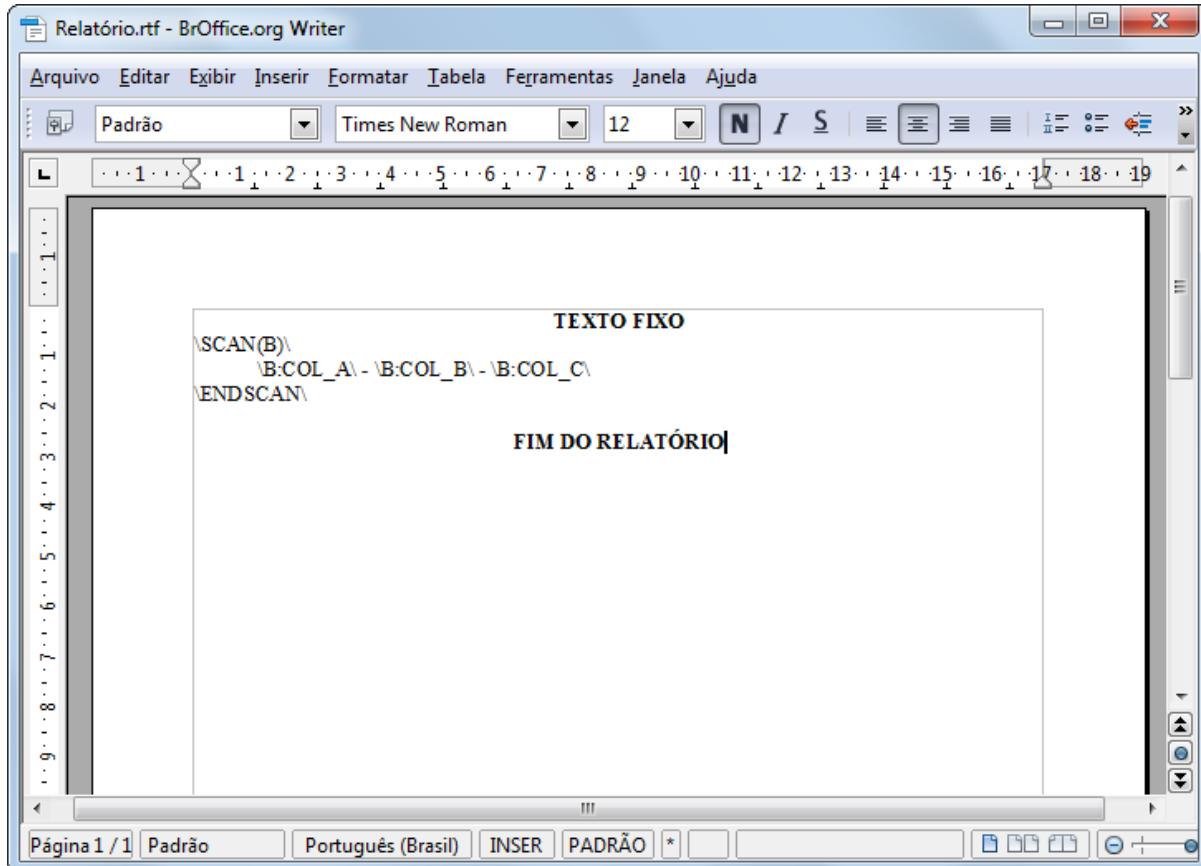


Salvando novo relatório

O leiaute do relatório RTF consiste de uma parte fixa (texto comum) e uma parte dinâmica

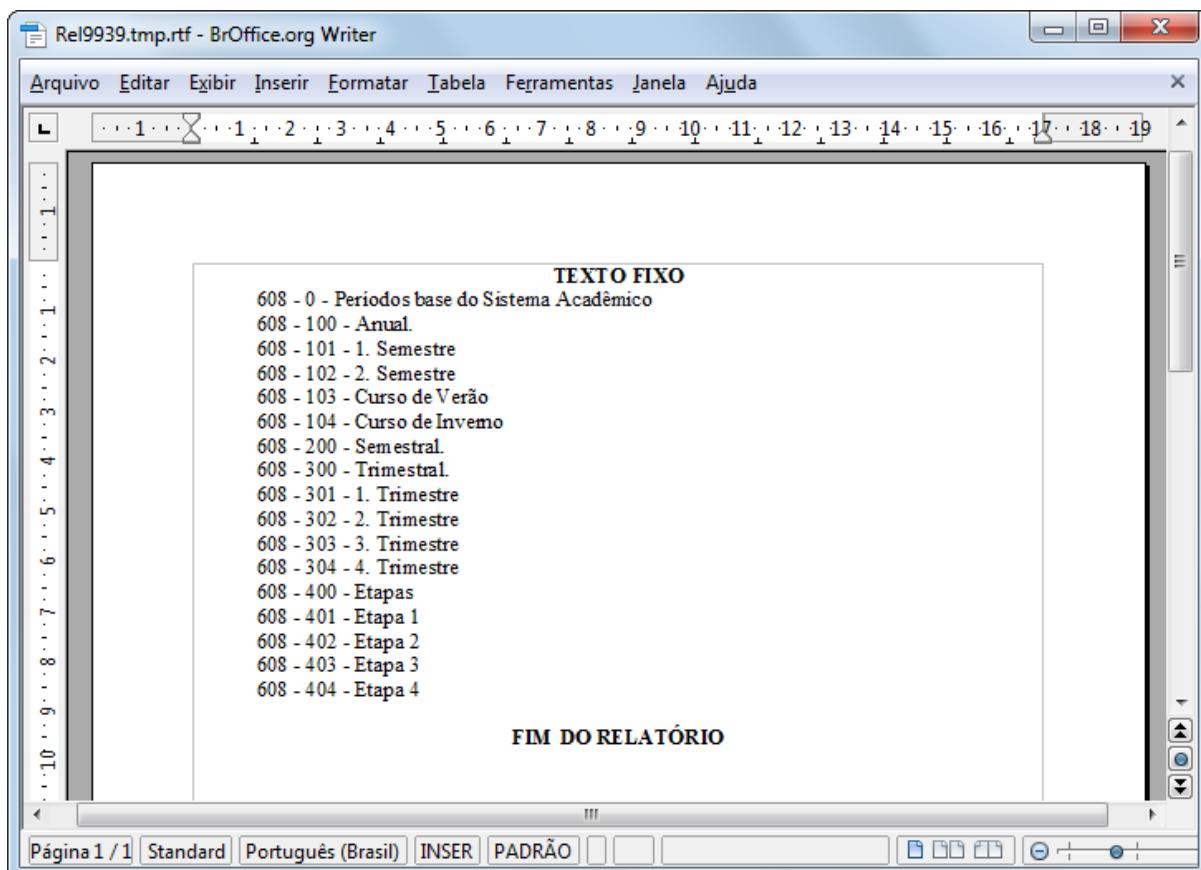
(código). A parte em código é interpretada e substituída por outra informação ou trabalhada para apresentar o resultado desejado.

Abaixo, um exemplo simples de relatório:



#### Primeiro relatório

O resultado pode ser visto abaixo:



#### Resultado do relatório

O que ocorreu é o seguinte: para cada registro retornado do banco pela consulta cadastrada, o relatório produziu uma linha correspondente. No caso, o texto fixo foi reproduzido no cabeçalho e no rodapé. Todo o código foi interpretado. No caso, código é tudo que foi escrito entre barras '\'.

**OBS:** Tudo o que for escrito no rtf entre barras (ex.: \informação) é considerado código. Se não for um comando reconhecido ou estiver errado, o texto do código será apresentado como foi escrito.

O comando básico usado é \SCAN(b)\ e \ENDSCAN\. Esses comandos iniciam e terminam a leitura de registros da consulta. Tudo o que estiver escrito entre esses dois comandos será igualmente repetido para cada registro retornado pela consulta do relatório. O que vier antes de \SCAN(b) pode ser considerado um "cabeçalho" e o que for escrito após o \ENDSCAN\ será como um "rodapé".

**OBS:** Os comandos \SCAN(b)\ e \ENDSCAN\ podem ser considerados um "loop" onde para cada registro retornado da consulta, o conteúdo entre os comandos será repetido.

**OBS:** Os comandos são aceitos em qualquer parte do relatório, inclusive antes do \SCAN(b)\ e após o \ENDSCAN\. Pode-se ter também vários \SCAN(b)\ e \ENDSCAN\ em sequência

ou aninhados.

Dentro do comando \SCAN(b)\ e \ENDSCAN\ foi usado os comandos \B:COL\_A\, \B:COL\_B\ e \B:COL\_C\. É através desses comandos que os valores da consulta são acessados. O comando padrão é \TABELA:[COLUNA]\, onde "TABELA" se refere à consulta do relatório em ordem alfabética crescente. "A" é a tabela PAR\_INSTITUICAO automaticamente adicionada ao relatório. A consulta cadastrada para o relatório é sempre "B". "COLUNA" é a coluna referenciada da consulta do relatório.

Não por acaso, \SCAN(B)\ é uma função, cujo parâmetro é a consulta "B". Nesse ponto, será executado o loop que irá percorrer os dados da consulta "B". Usando \B:COLUNA\ se acessa a informação em "COLUNA", retornado pela consulta B.

**OBS:** Para acessar a informação da consulta use \B:NOME\_COLUNA\.

Portanto, para cada comando "\B:COL\_A\ - \B:COL\_B\ - \B:COL\_C\" , temos a informação da coluna COL\_A, o texto "-", a informação da coluna COL\_B, outro "-" e a informação da coluna COL\_C.

### 5.2.5 Funções

As funções permitem processamento especial no relatório, podendo por exemplo acessar o nome do relatório, formatar data e hora, ter um valor por extenso e etc.

Funções podem ou não ter parâmetros, sendo que nesse caso, os parâmetros são colocados dentro de parênteses, separados por vírgula.

As funções são trechos de código, e como tal devem vir dentro de barras como por exemplo:

\FormatDateTime('DD/MM', Data)\

Os parâmetros podem ser texto (entre aspas), valores numéricos, variáveis ou colunas da consulta.

#### 5.2.5.1 Funções básicas

Funções podem ser utilizadas em qualquer parte do relatório, mas nem toda função faz sentido em qualquer parte. Por exemplo, não existe funcionalidade para a função "RecNo" no cabeçalho do relatório.

As funções podem ou não ter parâmetros. Mesmo que não tenham, deve-se adicionar parênteses "() " ao final do nome como em \ReportTitle()\.

Quando a função abaixo pede um valor tipo "Campo", deve-se informar uma coluna como "B:COL\_A" (sem as aspas). Se for "Texto", informa-se o texto com aspas. Se for Booleana, os

valores são TRUE ou FALSE.

Nome função	Descrição
ReportTitle()	Retorna o nome cadastrado para o relatório no sistema;
RecNo(Campo):Integer;	Retorna o número do registro atual da consulta;
GetRotulo(ID_Rótulo:Numérico):Texto	Retorna o conteúdo do rótulo, dado o número do rótulo;
FormatDateTime(Formato:Texto; Campo)	Retorna a data no formato do sistema ou do informado, se incluso. Ex: 'dd/mm', 'hh:mm';
IsNull(Campo):Booleana	Retorna True caso o valor do campo seja nulo;
Txt2rtf(Texto):Texto	Converte um texto para ser usado em RTF;
Odd (Num: Numerico)	Retorna True caso o valor informado seja par;
FormatFloat(Campo;Formato:Texto)	Retorna o número no formato #,##0.00 ou do formato informado, se incluso;
IsEqual(Valor1, Valor2):Booleana	Retorna True caso os argumentos (campo, número ou texto) informados sejam iguais.
RecordCount(Campo):Numérico	Retorna o número de registros da tabela;
GetNumeroPorExtenso(Número: Numérico):Texto	Retorna o número informado por extenso (Formato moeda);
IsLast(Campo):Booleana	Retorna True caso seja o último registro;
GetNumeroPorExtensoSemMoeda (Número: Numérico):Texto	Retorna o número informado por extenso;
Fimg (campo: Imagem; Tamanho: Numérico)	Formata a imagem no tamanho especificado;
Frftf(Campo)	Formata a informação RTF de uma coluna para ser visualizada corretamente no relatório.

As funções com retorno Booleano devem ser usadas em conjunto com a função IF, visto no próximo tópico.

Funções, por serem código, devem ser colocadas entre barras, como por exemplo "\ReportTitle()\\".

#### 5.2.5.2 Funções avançadas

##### Função IF

Uma das funções mais utilizadas em relatórios RTF é a função "IF". A função "IF" é análoga às encontradas em outras linguagens e permite a realização de testes, com a execução condicional de um bloco de texto e/ou comandos.

A função "IF" tem o seguinte formato:

\iff<Condição>\)

..textos e comandos se a condição for verdadeira

\else\

..textos e comandos se a condição for falsa

\endif\

Por exemplo, se fizermos o seguinte leiaute:

The screenshot shows the BrOffice.org Writer application window with the title bar 'Relatório.rtf - BrOffice.org Writer'. The menu bar includes 'Arquivo', 'Editar', 'Exibir', 'Inserir', 'Formatar', 'Tabela', 'Ferramentas', 'Janela', and 'Ajuda'. The toolbar has icons for document, page, and table orientation, followed by a ruler from 1 to 17. The main content area contains the following RTF code:

```
\REPORTTITLE()
TEXTO FIXO
\SCAN(B)\ 
  \IF(B:COL_B=0)\ 
    \B:COL_A\ - \B:COL_C\ 
  \ELSE\ 
    \B:COL_B\ - \B:COL_C\ 
  \ENDIF\ 
ENDSCAN\ 

FIM DO RELATÓRIO
```

### Uso da função IF e ReportTitle

Teremos o seguinte resultado:

The screenshot shows a Microsoft Word document window titled "RelDA56.tmp.rtf - BrOffice.org Writer". The menu bar includes Arquivo, Editar, Exibir, Inserir, Formatar, Tabela, Ferramentas, Janela, and Ajuda. The toolbar has icons for various document operations. The main content area contains the following text:

**Relatório RTF**

**TEXTO FIXO**

**608 - Períodos base do Sistema Acadêmico**

- 100 - Anual.
- 101 - 1. Semestre
- 102 - 2. Semestre
- 103 - Curso de Verão
- 104 - Curso de Inverno
- 200 - Semestral.
- 300 - Trimestral.
- 301 - 1. Trimestre
- 302 - 2. Trimestre
- 303 - 3. Trimestre
- 304 - 4. Trimestre

**609 - Tipos de pré-requisitos**

- 1 - Disciplina
- 2 - Co-requisito
- 3 - Carga horária vencida
- 4 - Créditos vencidos
- 5 - Período vencido
- 7 - Frequencia

Resultado obtido pela função IF

Conforme pode ser visto na imagem do resultado, toda vez que o valor da coluna "COL\_B" for zero, é impresso a linha em negrito. Caso contrário, é impresso a linha sem negrito. Também pode ser visto o resultado da função ReportTitle, que retorna a descrição da aplicação relatório cadastrado na árvore do SI\*.

### Função CheckGroup

Outra função utilizada em relatórios é a CheckGroup. Com essa função, podemos determinar se o valor de uma coluna mudou de um registro para outro, permitindo que, junto com a função IF, criemos "agrupamentos" como o que ocorre no exemplo acima. No caso anterior, havia uma facilidade que é o valor zero para o item que deixamos em negrito. Mas se quisermos agrupar

testando se o valor de "COL\_A" mudar (no caso acima, de 608 para 609), teremos que usar a função.

A função tem a seguinte assinatura:

CHECKGROUP(Campo: Texto, Variável: Texto, Reset: Booleano): Booleano

onde:

**Campo:** É o valor a ser testado com o valor anterior.

**Variável:** É o local que fica armazenado o último valor. Usado para comparar com "Campo" atual.

**Reset:** Reinicializa o valor da variável. Mantenha sempre TRUE, a menos que exista dois CheckGroup aninhados para a mesma variável.

e o resultado é um valor booleano que pode ser usado junto a um IF. Essa função deve ser usada como a seguir:

Relatório.rtf - BrOffice.org Writer

Arquivo Editar Exibir Inserir Formatar Tabela Ferramentas Janela Ajuda

1 ··· 1 ··· 2 ··· 3 ··· 4 ··· 5 ··· 6 ··· 7 ··· 8 ··· 9 ··· 10 ··· 11 ··· 12 ··· 13 ··· 14 ··· 15 ··· 16 ··· 17 ··· 18 ··· 19 ···

\REPORTTITLE\

**TEXTO FIXO**

\SCAN(B)\  
  \IF(CHECKGROUP(B:COL\_A, 'TEMP', TRUE))\

\TABELA: \B:COL\_A\ - \B:COL\_C\

\ELSE\

\B:COL\_B\ - \B:COL\_C\

\ENDIF\

\ENDSCAN\

**FIM DO RELATÓRIO**

**Função Checkgroup**

Com o seguinte resultado:

RelDAF8.tmp.rtf - BrOffice.org Writer

Arquivo Editar Exibir Inserir Formatar Tabela Ferramentas Janela Ajuda

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

**Relatório RTF**

**TEXTO FIXO**

**TABELA: 608 - Periodos base do Sistema Acadêmico**

100 - Anual.  
101 - 1. Semestre  
102 - 2. Semestre  
103 - Curso de Verão  
104 - Curso de Inverno  
200 - Semestral.  
300 - Trimestral.  
301 - 1. Trimestre  
302 - 2. Trimestre  
303 - 3. Trimestre  
304 - 4. Trimestre

**TABELA: 609 - Tipos de pré-requisitos**

1 - Disciplina  
2 - Co-requisito  
3 - Carga horária vencida

Página 1 / 1 Padrão Padrão INSER PADRÃO \*

**Resultado da função Checkgroup**

O que aconteceu é o seguinte: O relatório imprime o cabeçalho. Então entra na parte do SCAN(B). Ao passar pelo primeiro registro, a função CheckGroup compara o valor de COL\_A com o valor da variável TEMP. Se for diferente, então é um novo grupo e ela retorna TRUE para a função IF.

Para o primeiro registro, o valor em TEMP é " (vazio) e o valor em COL\_A é 608. A função retorna TRUE e guarda o valor de COL\_A em TEMP. Para os registros seguintes, TEMP tem 608 e COL\_A continua 608, fazendo com que a função CheckGroup retorne FALSE (itens 100 ao 304). Para o registro onde COL\_A é 609, a função compara TEMP (com 608) e o valor em COL\_A, retornando TRUE novamente. Dessa forma aparece o primeiro item com COL\_A 609 em negrito conforme imagem acima.

### 5.2.5.3 Somatórios

As vezes é necessário mostrar um contador ou somatório de uma informação da coluna. Para tal usamos funções para somar e contar específicas. Para realizar essa tarefa, precisamos fazer uso de variáveis.

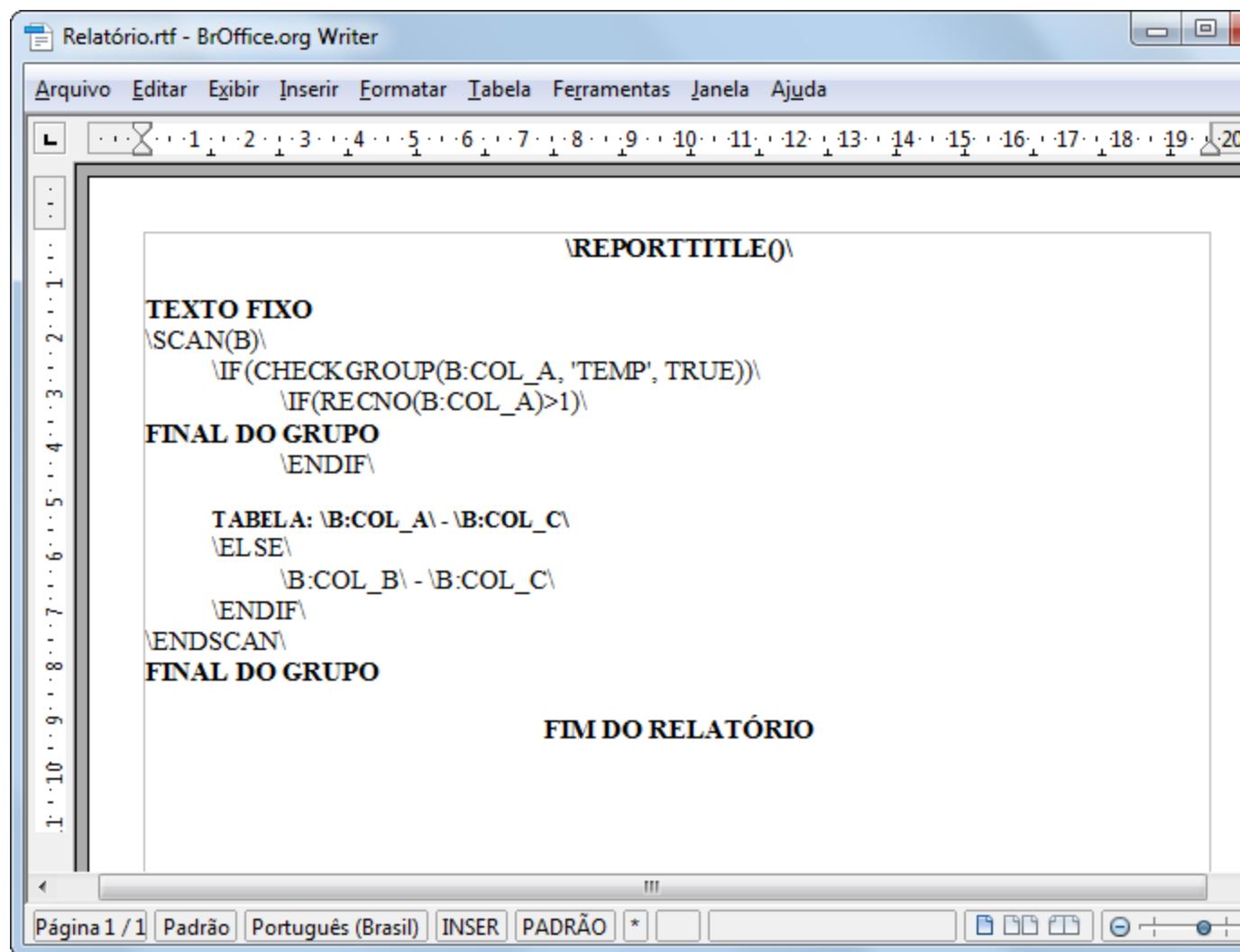
Variáveis servem para armazenar valores, possibilitando a criação de relatórios mais complexos.

Existem as seguintes funções nos relatórios RTF para trabalhar com variáveis:

Nome função	Descrição
VAR(Variável)	Cria nova variável.
SET(Valor, Variável)	Atribui Valor à variável.
\Variável\	Mostra o conteúdo da variável.
RESET(Variável)	Inicializa a variável com zero/vazio.
SUM(Valor, Variável)	Acrescenta à "Variável", o conteúdo de "Valor".
CTN(Campo, Variável)	Conta o número de valores numéricos diferentes de zero.
CTS(Campo, Variável)	Conta o número de valores texto diferentes de vazio.

Para usar contador/somatórios, é necessário (por lógica) criar uma área de rodapé para o agrupamento de forma a mostrar um total por grupo. Nessa área, será mostrado o valor do contador. Este valor, por sua vez, será obtido a medida que as informações de cada registro forem sendo impresso no relatório.

Segue o seguinte exemplo:



Exemplo de layout com rodapé

Note que foi acrescentado código para o rodapé antes do texto usual. O código na verdade imprime o rodapé do grupo anterior, antes da informação do novo grupo ser impresso. O "IF" impede que o rodapé seja impresso para o primeiro agrupamento (COL\_A 608). Note também a repetição do rodapé após o "ENDSCAN". Esse é o rodapé do último grupo.

O resultado pode ser visto abaixo:

The screenshot shows a document titled "Relatório RTF" in BrOffice.org Writer. The content includes:

- TEXTO FIXO**
- TABELA: 608 - Períodos base do Sistema Acadêmico**
  - 100 - Anual.
  - 101 - 1. Semestre
  - 102 - 2. Semestre
  - 103 - Curso de Verão
  - 104 - Curso de Inverno
  - 200 - Semestral.
  - 300 - Trimestral.
  - 301 - 1. Trimestre
  - 302 - 2. Trimestre
  - 303 - 3. Trimestre
  - 304 - 4. Trimestre
- FINAL DO GRUPO**
- TABELA: 609 - Tipos de pré-requisitos**
  - 1 - Disciplina
  - 2 - Co-requisito

The bottom of the screen shows the standard Writer toolbar and status bar indicating "Página 1 / 1".

**Relatório com rodapé**

Agora que temos um rodapé, vamos adicionar um número de registros. Veja o código a seguir:

The screenshot shows a window titled "Relatório.rtf - BrOffice.org Writer". The menu bar includes Arquivo, Editar, Exibir, Inserir, Formatar, Tabela, Ferramentas, Janela, and Ajuda. The toolbar has icons for text style, font, size, and other document functions. The main content area contains RTF code:

```
\REPORTTITLE()  
TEXTO FIXO  
\SCAN(B)\  
  \IF(CHECKGROUP(B:COL_A, 'TEMP', TRUE))\  
    \IF(RECNO(B:COL_A)>1)\  
      FINAL DO GRUPO. Total:\TOTAL\RESET(TOTAL)\  
    \ENDIF\  
  \ELSE\  
    \B:COL_B\ - \B:COL_C\  
  \ENDIF\  
\ENDSCAN, CTN(B:COL_B, TOTAL)\  
FINAL DO GRUPO. Total:\TOTAL\  
FIM DO RELATÓRIO
```

The status bar at the bottom indicates "Leiaute com contador".

No caso, ocorre de usarmos a coluna COL\_B, visto que para o caso da tabela estruturada, o item zero é a descrição da tabela. A função CTN não levará em consideração o valor zero e teremos o número correto de itens.

O uso dessas funções é diferente do processo usual. Elas devem ser colocadas junto ao comando ENDSCAN. O comando padrão é o seguinte:

```
\ENDSCAN, função 1, função 2, função N\
```

Ao fim de cada loop do comando SCAN, as funções serão executadas.  
O resultado do leiaute pode ser visto abaixo:

The screenshot shows a document titled "Rel4454.tmp.rtf - BrOffice.org Writer". The menu bar includes Arquivo, Editar, Exibir, Inserir, Formatar, Tabela, Ferramentas, Janela, and Ajuda. The toolbar has icons for text, bold, italic, underline, etc. The main content area contains:

**Relatório RTF**

**TEXTO FIXO**

**TABELA A: 608 - Períodos baseado Sistema Acadêmico**

100 - Anual.
101 - 1. Semestre
102 - 2. Semestre
103 - Curso de Verão
104 - Curso de Inverno
200 - Semestral.
300 - Trimestral.
301 - 1. Trimestre
302 - 2. Trimestre
303 - 3. Trimestre
304 - 4. Trimestre
400 - Etapas
401 - Etapa 1
402 - Etapa 2
403 - Etapa 3
404 - Etapa 4

**FINAL DO GRUPO. Total: 16**

**TABELA A: 609 - Tipos de pré-requisitos**

1 - Disciplina
2 - Co-requisito
3 - Carga horária vencida
4 - Cadastrado em sistema

At the bottom, there are buttons for Página 1 / 1, Padrão, INSER, PADRÃO, and a search bar. The status bar at the bottom says "Resultado com contador de registros".

O uso da função SUM é análoga a CTN.

Um contador geral (sem ser por grupo), tem basicamente o mesmo código. A diferença é a falta do comando RESET ao final do grupo. O local ideal para a informação é após o último grupo. Dessa forma teremos um contador/somador para o relatório inteiro.

#### 5.2.5.4 Outras funções

Existem ainda algumas funcionalidades possíveis para se criar no leiaute. Seguem abaixo:

**Cabeçalho e Rodapé:** Podemos usar as funções de cabeçalho e rodapé para inserir um texto padrão em todas as páginas. Normalmente usado para mostrar o ícone da instituição, nome do relatório, data, hora e etc. No rodapé, é normalmente mostrado número de página.

**Quebra de página:** As vezes é interessante não apenas criar um agrupamento no relatório,

conforme tópico anterior, mas também forçar uma quebra no relatório de forma que o novo grupo inicie em uma nova página. Para isso basta ir nas opções do editor e inserir uma quebra de página. O local apropriado seria após o código do rodapé.

Veja o código:

The screenshot shows the BrOffice.org Writer application window titled "Relatório.rtf - BrOffice.org Writer". The menu bar includes Arquivo, Editar, Exibir, Inserir, Formatar, Tabela, Ferramentas, Janela, and Ajuda. The toolbar includes icons for document, page, and table orientation, and page numbers 1 through 11. The main content area contains RTF code:

```
\REPORTTITLE()
TEXTO FIXO
SCAN(B)
\IF(CHECKGROUP(B:COL_A,'TEMP',TRUE))
\IF(RECNO(B:COL_A)>1)
FINAL DO GRUPO. Total: \TOTAL\RESET(TOTAL)

TABELA: \B:COL_A\ - \B:COL_C\
\ELSE
\B:COL_B\ - \B:COL_C\
\ENDIF
ENDSCAN, CTN(B:COL_B, TOTAL)
FINAL DO GRUPO. Total: \TOTAL\

FIM DO RELATÓRIO
```

The left margin shows page numbers from 0 to 19. The bottom navigation bar includes buttons for Página 2 / 2, Padrão, Português (Brasil), INSER, PADRÃO, and other document control buttons.

Leiaute com quebra de página

E o resultado:

The screenshot shows a document titled "Relatório RTF" in BrOffice.org Writer. The menu bar includes Arquivo, Editar, Exibir, Inserir, Formatar, Tabela, Ferramentas, Janela, and Ajuda. The toolbar below the menu has icons for text, tables, and other document functions. A horizontal ruler at the top shows page numbers 1 through 19. The main content area contains several sections of text:

**TEXTO FIXO**

**TABELA: 608 - Períodos base do Sistema Acadêmico**

100 - Anual.  
101 - 1. Semestre  
102 - 2. Semestre  
103 - Curso de Verão  
104 - Curso de Inverno  
200 - Semestral.  
300 - Trimestral.  
301 - 1. Trimestre  
302 - 2. Trimestre  
303 - 3. Trimestre  
304 - 4. Trimestre

**FINAL DO GRUPO.** Total: 11

**TABELA: 609 - Tipos de pré-requisitos**

1 - Disciplina  
2 - Co-requisito  
3 - Carga horária vencida  
4 - Créditos vencidos  
5 - Período vencido  
7 - Frequência

**FINAL DO GRUPO.** Total: 6

**FIM DO RELATÓRIO**

The bottom of the window shows navigation buttons: Página 1 / 2, Padrão, INSER, PADRÃO, and a series of icons for file operations like Open, Save, Print, and Close. The status bar at the bottom right indicates "PÁGINA 1 DE 1".

Relatório com quebra de página

# **Top Level Intro**

This page is printed before a new  
top-level chapter starts

**Part**



**VI**

## 6 Apêndice A

### 6.1 SGCPack

SGCPack é um vetor de 14 (de zero a 13) posições que deve ser passado em TODAS as chamadas de métodos servidores (geralmente possui o nome de CtxSGCA na assinatura dos métodos).

Ele é utilizado para fins de segurança, como por exemplo, garantir que as chamadas aos métodos foram feitas de dentro do SIM ou de aplicações autorizadas. É utilizado também para aplicar restrições de dados, visto que possui as informações do usuário e da aplicação.

Valores em cada posição do vetor:

**0** = IdUsuario que está utilizando a aplicação e consequentemente fazendo a chamada do método.

**1** = IdAplicacao que está fazendo a chamada do método

**2** = IdUsuario + IdAplicacao + Nome da servidora + Endereço ip da máquina cliente criptografados.

**3** = Endereço IP da máquina cliente.

**4** = Nome da servidora.

**5** = Endereço IP do servidor.

**6** = RESERVADO.

**7** = Tipo de retorno das consultas enviadas ao banco. Possíveis valores: **M** (Midas, que é o padrão) ou **X** (XML).

**8** = Indica se o contexto deve aplicar as restrições de dados, caso existam. Possíveis valores: **S** ou **N**.

**9** = Senha do usuário criptografada;

**10** = Informações extras (não utilizado).

**11** = Alias do Banco de dados.

**12** = Indica o tipo de aplicação onde será utilizado o CtxSGCA. Possíveis valores: **W** (web) ou **F** (normal).

**13** = Indica se o usuário é um superusuário. Por exemplo o ADM. Possíveis valores: **S** ou **N**

Existe uma constante cadastrada para cada posição na unit **uBsConsts**. Nesse caso não é necessário decorar cada posição para quando for preciso utilizar. Por exemplo, se for necessário utilizar o endereço ip da máquina cliente (posição 3) dentro de algum método, ao invés de utilizar CtxSGCA[3], pode ser utilizado CtxSGCA[POS\_IP\_CLIENTE].

Abaixo a lista das constantes que estão em **uBsConsts**:

```
POS_ID_USUARIO = 0;  
POS_APPLIC_CLIENTE = 1;  
POS_ENCRYPTED = 2;  
POS_IP_CLIENTE = 3;  
POS_APPLIC_SERVIDORA = 4;  
POS_IP_SERVIDORA = 5;  
POS_RESERVED = 6;  
POS_TIPO_RETORNO_CTX = 7;  
POS_IND_USAR_RESTR = 8;  
POS_PW_USUARIO = 9;  
POS_EXTRA_INFO_CTX = 10;  
POS_ALIAS = 11;  
POS_TIPO_CLIENTE = 12;  
POS_IND_SUPER_USUARIO = 13;
```

O CtxSGCA é montado antes da chamada do método servidor dentro do *TCliBusiness* da aplicação cliente.

Por exemplo: No código abaixo, existe um método na servidora **pjrTributacao**, interface **IContribuinteImob**, que retorna se um determinado contribuinte possui CPF ou CNPJ cadastrado. O CtxSGCA é montado no trecho de código em vermelho:

Declaração do método em prjTributacao:

```
function ContribuintePossuiCPFouCNPJ( IdContribuinte: Integer;
                                         CtxSGCA: OleVariant; const CtxDb: IBsContextoDb; var Errors: ICpErrors): WordBool; safecall;
```

Chamada do método na aplicação cliente:

```
function TCliContribuinteImob.ContribuintePossuiCPFouCNPJ( IdContribuinte: Integer): WordBool;
var Errors: ICpErrors;
begin
    result := (RemoteBusiness as IContribuinteImobDisp).ContribuintePossuiCPFouCNPJ(IdContribuinte, GetCtxSGCA, nil, Errors);
    TrataCpErrors(Errors);
end;
```

# **Top Level Intro**

This page is printed before a new  
top-level chapter starts

**Part**

**VIII**

## 7 Apêndice B

### 7.1 TfrCPD

Formulário base para criação de aplicações SI\*. Possui algumas informações chaves como IdUsuario e IdAplicacao.

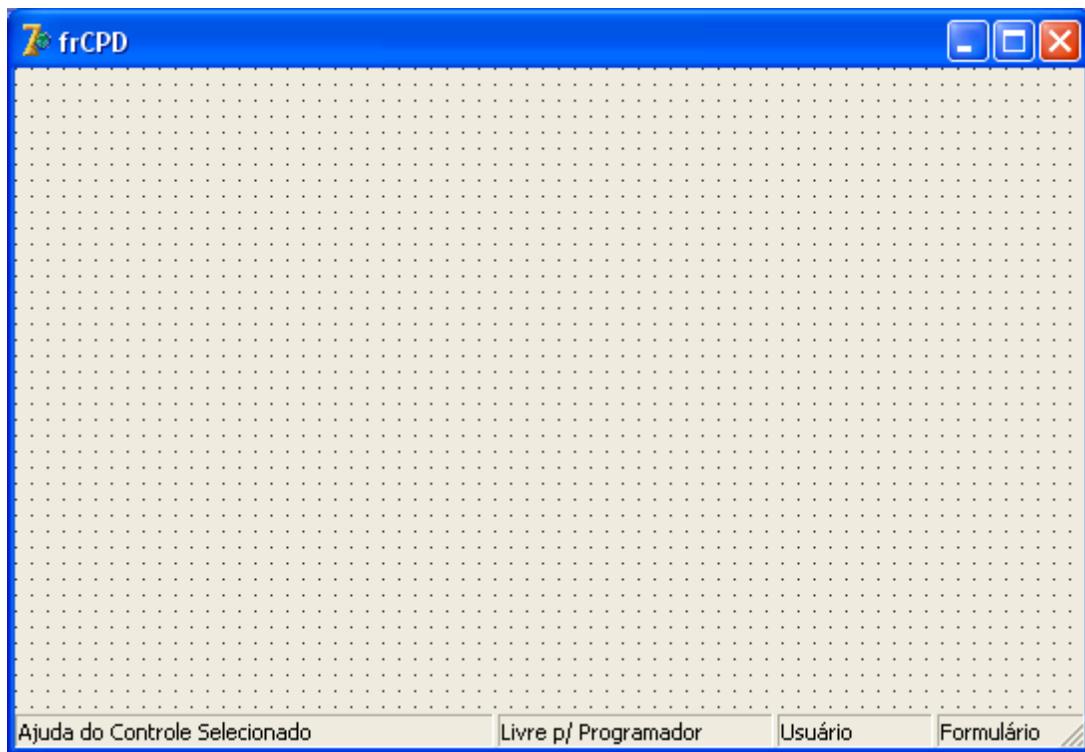
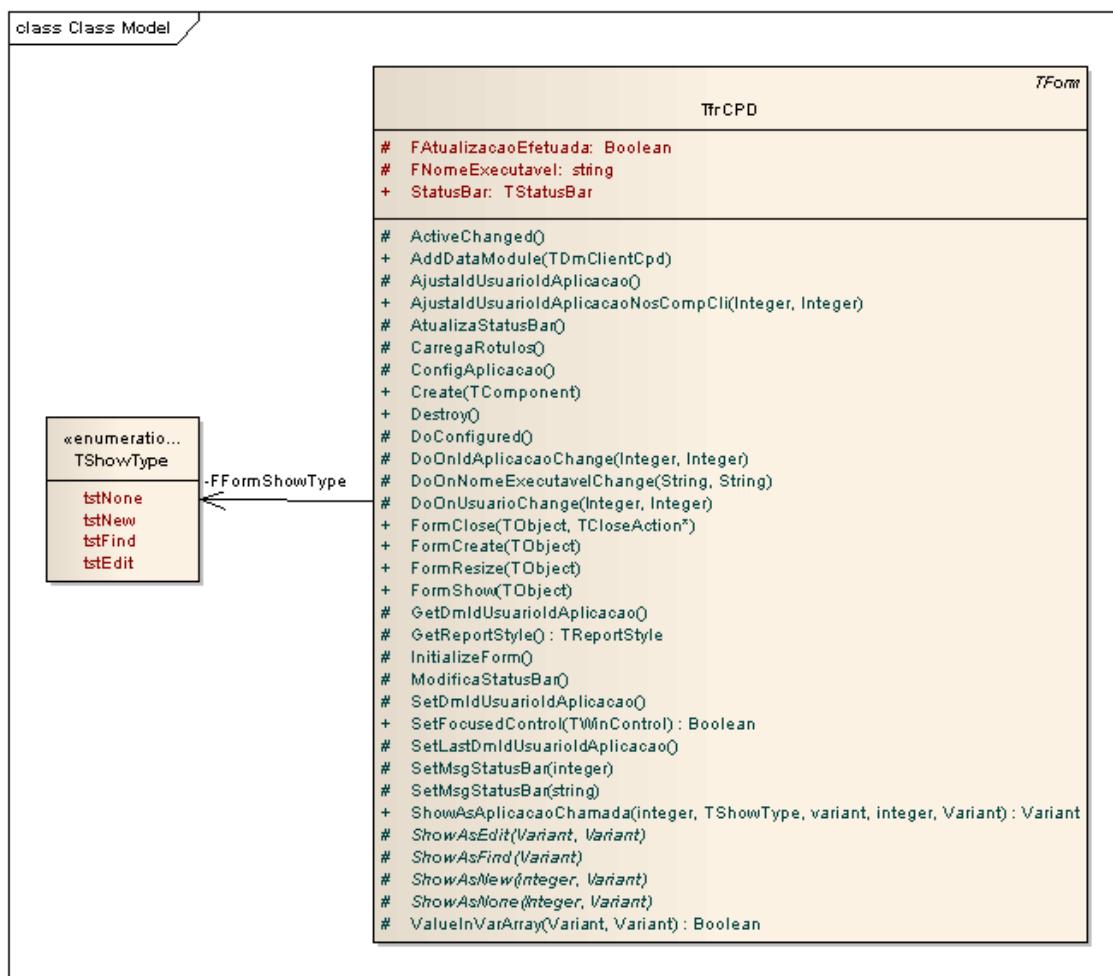


Diagrama de Classes:



### 7.1.1 Métodos Virtuais

Podem ser reescritos nas classes descendentes, desde que estejam na seção **protected** e com a palavra reservada **override**.

Nesta seção serão descritos os métodos mais utilizados.

#### 7.1.1.1 CarregaRotulos

```
procedure CarregaRotulos;
```

Utilizado para carregar os rótulos dos controlos, descrição dos campos e nomes de colunas.

### 7.1.1.2 InitializeForm

```
procedure InitializeForm;
```

Método chamado logo após o formulário ter sido criado.

### 7.1.1.3 ShowAsAplicacaoChamada

```
function ShowAsAplicacaoChamada(IdAplicChamada: integer; ShowType: TShowType; LKValues: variant; ForeignValue: integer; Reserved: Variant): Variant;
```

Método usado quando se quer mostrar o formulário como se estivesse chamando uma outra aplicação. (ver métodos abaixo: ShowAsNew, ShowAsEdit e ShowAsNone

Parâmetros:

**ShowType** - **tstNew**(ShowAsNew), **tstEdit**(ShowAsEdit), **tstNone**(ShowAsNone), **tstFind**(ShowAsFind).

**LKValues** - chave lógica do registro a ser editado (só se aplica se ShowType for **tstEdit**).

**ForeignValue** - valor da chave física do registro que está sendo manipulado na aplicação chamada (só se aplica se ShowType for **tstNew**)

### 7.1.1.4 ShowAsEdit

```
procedure ShowAsEdit(LkValues, Reserved: Variant);
```

Deve ser sobreescrito quando o formulário precisa ser chamado de outra aplicação com a finalidade de alterar um registro. O parâmetro **LkValues** deve conter a chave lógica do registro a ser alterado. O código deste método normalmente é semelhante ao código executado no "Localizar", pois ele deve posicionar o registro passado em **LkValues** na tela.

### 7.1.1.5 ShowAsNew

```
procedure ShowAsNew(ForeignValue: integer; Reserved: Variant);
```

Este método deve ser sobreescrito quando o formulário precisa ser chamado de outra aplicação com a finalidade de incluir um registro. O parâmetro **ForeignValue** deve conter a chave física do registro que está sendo manipulado pela aplicação chamadora. Por exemplo, se aplicação chamadora for o Cadastro de Contribuintes e a aplicação chamada for o Cadastro Mobiliário, então o parâmetro **ForeignValue** deverá conter o **IdContribuinte**.

### 7.1.1.6 ShowAsNone

```
procedure ShowAsNone(ForeignValue: Integer; Reserved: Variant);
```

Deve ser sobreescrito quando o formulário precisa ser chamado de outra aplicação com a finalidade de mostrar um registro.

## 7.1.2 Propriedades

Podem ser acessadas a partir de qualquer formulário. Nessa seção serão descritas as propriedades mais utilizadas.

### 7.1.2.1 RemoteSGCA

```
property RemoteSGCA: ISGCADisp
```

Retorna uma referência para a interface ISGCA. A partir dessa referência podem ser invocados os métodos que estão declarados em ISGCA.

### 7.1.2.2 ShowingAsNew

```
property ShowingAsNew: Boolean
```

Quando o formulário foi chamado para inserir um registro novo, esta propriedade é True, senão é False.

### 7.1.2.3 ShowingAsEdit

```
property ShowingAsEdit: Boolean
```

Quando o formulário foi chamado para editar um registro novo, esta propriedade é True, senão é False.

### 7.1.2.4 NomeExecutavel

```
property NomeExecutavel: String
```

Retorna o nome do executável (.exe) da aplicação.

### 7.1.2.5 AtualizacaoEfetuada

```
property AtualizacaoEfetuada: Boolean
```

Propriedade auxiliar para ser utilizada para controlar se o usuário alterou algum registro.

Por exemplo: Clicou em Alterar e depois Salvar: AtualizacaoEfetuada = True.

Clicou em Alterar e depois Cancelar: AtualizacaoEfetuada = False.

### 7.1.2.6 IdAplicacao

```
property IdAplicacao: integer
```

Retorna ou define o valor de IdAplicacao da aplicação que está sendo executada.

#### 7.1.2.7 IdUsuario

```
property IdUsuario: integer
```

Retorna ou define o valor de IdUsuario (CodOperador) que está manipulando a aplicação.

#### 7.1.2.8 IndAuditada

```
property IndAuditada: Boolean
```

Retorna se a aplicação é auditada. Se for auditada cada vez que a aplicação for aberta ou fechada uma linha será incluída na tabela *USUARIOS\_ATIVOS*.

## 7.2 TfrCPDTool

Formulário herdado de TfrCPD. Possui barra de ferramentas com botões Novo, Alterar, Excluir, Localizar e Imprimir.

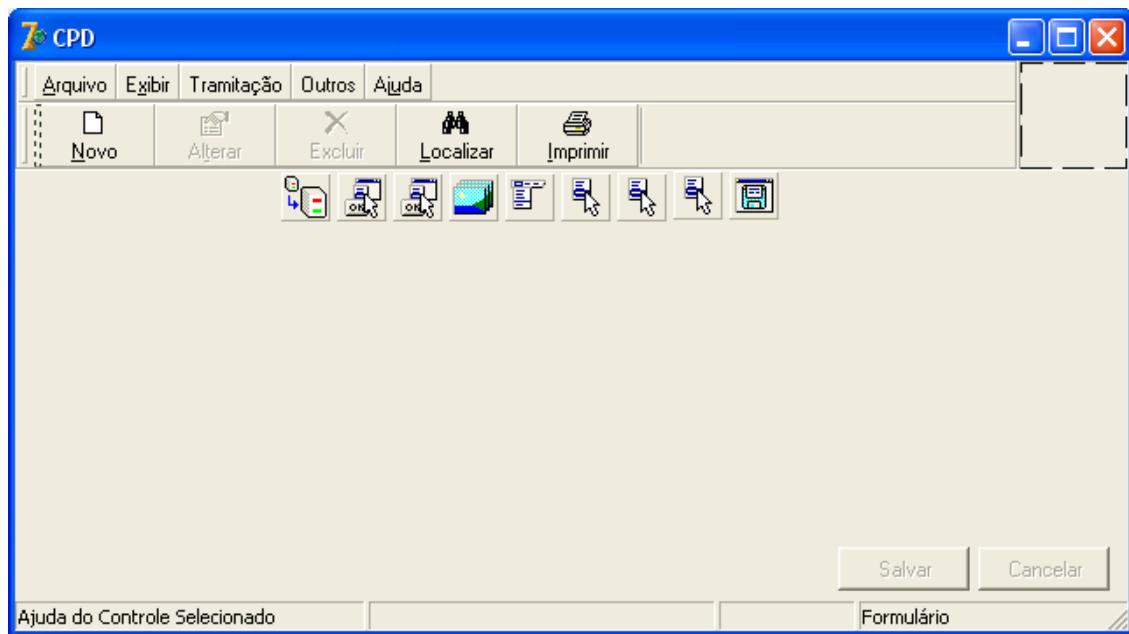
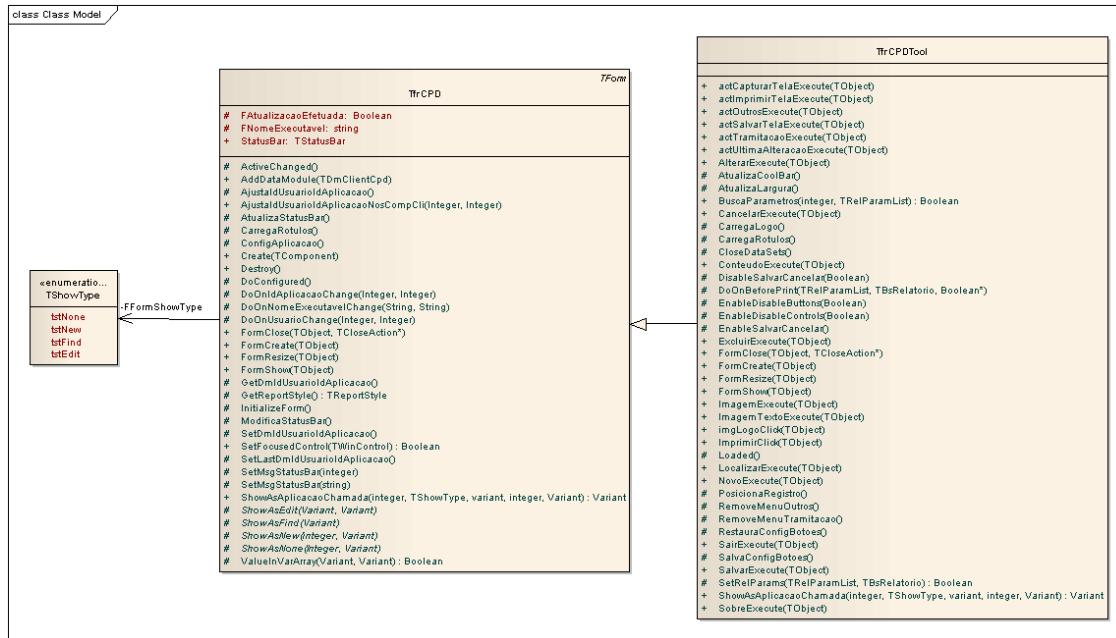


Diagrama de Classes:



## 7.2.1 Métodos Virtuais

Podem ser reescritos nas classes descendentes, desde que estejam na seção **protected** e com a palavra reservada **override**.

### 7.2.1.1 CloseDataSets

```
procedure CloseDataSets;
```

Fecha os datasets de CliBusiness auxiliares não mais necessários caso o usuário cancele a inclusão de um registro ou apague um registro.

Exemplo:

```
procedure TfrmCPDTool.CloseDataSets;
begin
    CliSimulacaoCalculo.CliDsGetRecord.Close;
    CliTributoImp.CliDsGetRecord.Close;
    CliSimulacaoEmissao.GetCdsGetSimulacoesByIdEmissao.Close;
    CliVersaoCalculo.GetCdsGetVariaveisVersao.Close;
end;
```

### 7.2.1.2 EnableDisableControls

```
procedure EnableDisableControls(aEnable: Boolean)
```

Faz a habilitação dos controles de edição de registro.

Esse método é chamado em TfrCPDTool nessas situações:

- FormShow (Passado False como parâmetro)
- NovoExecute (True)
- SalvarExecute (False)
- CancelarExecute (False)
- AlterarExecute (True)

Exemplo:

```
procedure TfrCPDTool.EnableDisableControls(aEnable: Boolean)
begin
  BsEditFantasia.Enabled := aEnable;
  LabelFantasia.Enabled := aEnable;
  BsDTNascimento.Enabled := aEnable;
end;
```

#### 7.2.1.3 PosicionaRegistro

```
procedure PosicionaRegistro;
```

Posiciona os *CliBusiness* auxiliares, geralmente as chaves estrangeiras, após clicar em localizar ou o em cancelar. Pode ser utilizado também para trazer dados adicionais referente ao registro que foi localizado.

Exemplo: Aplicação de cadastro de contribuintes, possui a tabela *PESSOAS* como chave estrangeira, e deseja-se mostrar o nome da pessoa na tela. Além disso, deseja-se mostrar todas as atividades mobiliárias do contribuinte:

```
procedure TfrCPDTool.PosicionaRegistro;
begin
  if not CliContribuinteImob.FieldByName('ID_PESSOA').IsNull then
    CliPessoal.GetRecordPK(CliContribuinteImob.FieldByName('ID_PESSOA').AsInteger, False, False);
    CliContribuinteImob.GetCadastradosEconomicos(CliContribuinteImob.PKfield.AsInteger);
end;
```

#### 7.2.1.4 SetRelParams

```
function SetRelParams(RelParamList: TRelParamList; BsRelatorio: TBsRelatorio): Boolean;
```

Informa os valores dos parâmetros para o relatório. Deve ser sobreescrita quando os parâmetros de um relatório não estiverem no *CliBusiness* informado na propriedade *CliPrincipal*.

Exemplo:

```
function TfrCPDTool.SetRelParams(RelParamList: TRelParamList; BsRelatorio: TBsRelatorio): Boolean;
begin
  with dmClientCPD do
    BsRelatorio.AddParam('ID_TIPO_DOC', '', CliDocumento.FieldByName('ID_TIPO_DOC').AsInteger);
  Result := True;
end;
```

## 7.2.2 Propriedades

Podem ser acessadas a partir de qualquer formulário. Nessa seção serão descritas as propriedades mais utilizadas.

### 7.2.2.1 FrmState

```
property FrmState: TFrmState
```

Retorna o estado do formulário. Possíveis valores:

**fstBrowse** - Visualizando algum registro na tela.

**fstEdit** - Foi clicado no botão "Editar" para alteração de algum registro.

**fstInsert** - Foi clicado no botão "Novo" para a inclusão de algum registro.

**fstNone** - Nenhum registro posicionado na tela.

# **Top Level Intro**

This page is printed before a new  
top-level chapter starts

**Part**

**VIII**

## 8 Apêndice C

### 8.1 UtilCPD

Biblioteca com rotinas auxiliares que pode ser utilizada em qualquer aplicação Cliente.

OBS: essa biblioteca **NÃO** deve ser utilizada em aplicações servidores e relatórios. Para essa finalidade utilize a biblioteca *UtilCPDSrv*

#### 8.1.1 Principais Métodos

##### 8.1.1.1 AcrescentaZerosEsq

```
function AcrescentaZerosEsq(const str: String; TamanhoTotal: Integer): String;
```

Adiciona zeros a esquerda da variável **str** passada, até que fique do tamanho da variável passada **TamanhoTotal**

##### 8.1.1.2 CheckCGC

```
function CheckCGC(Value: String): Boolean;
```

Verifica se o CNPJ passado por parâmetro é válido, inclusive calculando se o dígito verificador está correto. Pode ser passado CNPJ com máscara (99.999.999/9999-99) ou sem máscara.

##### 8.1.1.3 CheckCPF

```
function CheckCPF(Value: String): Boolean;
```

Verifica se o CPF passado por parâmetro é válido, inclusive calculando se o dígito verificador está correto. Pode ser passado CPF com máscara (999.999.999-99) ou sem máscara.

##### 8.1.1.4 CheckPIS

```
function CheckPIS(Value: String): Boolean;
```

Verifica se o PIS passado por parâmetro é válido. O verificação da validade é feita através do cálculo "módulo 11".

##### 8.1.1.5 FormatDateCpd

```
function FormatDateCpd(aDateTime: TDateTime): string;
```

Retorna uma string com a data passada por parâmetro formatada no padrão utilizado pelo

SIM.

Formato de retorno: '**dd/mm/yyyy**'

#### 8.1.1.6 FormatDateTimeCpd

```
function FormatDateTimeCpd(aDateTime: TDateTime): string;
```

Retorna uma string com a data e hora passada por parâmetro formatada no padrão utilizado pelo SIM.

Formato de retorno: '**dd/mm/yyyy' 'hh:nn:ss**'

#### 8.1.1.7 FormatTimeCpd

```
function FormatTimeCpd(aDateTime: TDateTime): string;
```

Retorna uma string com a hora passada por parâmetro formatada no padrão utilizado pelo SIM.

Formato de retorno: '**hh:nn:ss**'

#### 8.1.1.8 GetBsContextoDb

```
function GetBsContextoDb: IBsContextoDb;
```

Instancia um objeto TBsContextoDb.

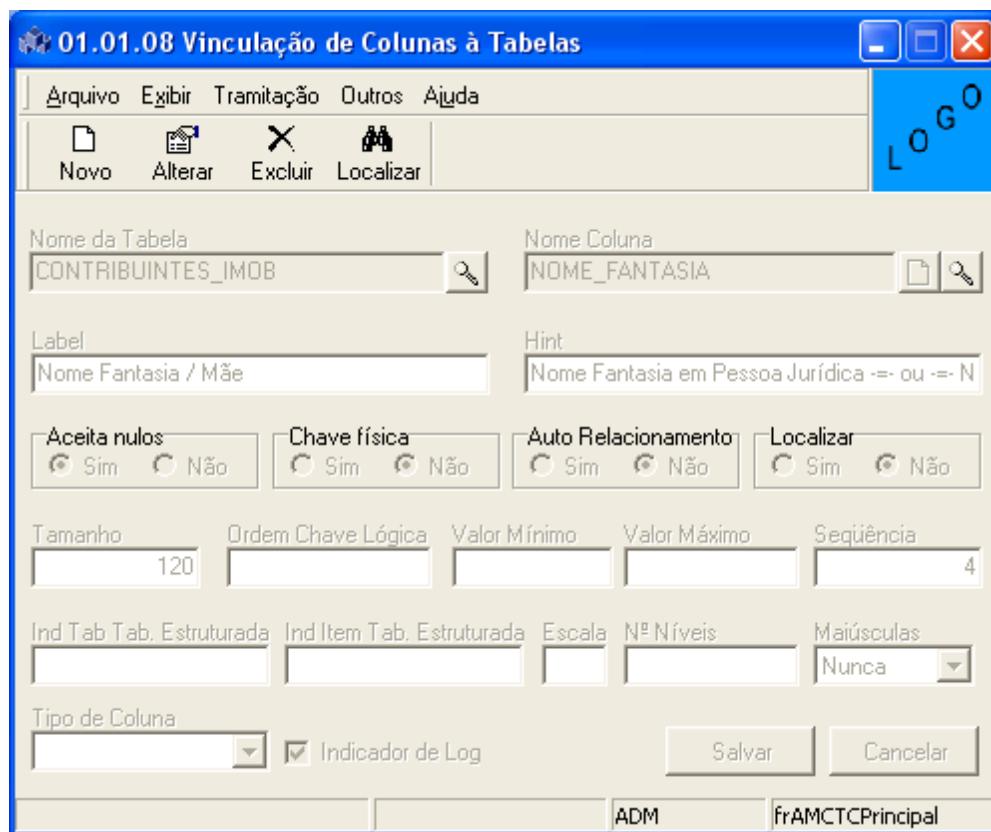
#### 8.1.1.9 GetDescrTabCol

```
function GetDescrTabCol(const NomeTabela, NomeColuna: WideString): WideString;
```

Retorna a descrição da coluna que foi previamente cadastrado na aplicação **AMCTabela.exe**.

Utilizado nos rótulos dos controles das aplicações clientes.

Exemplo:



```
LabelFantasia.Caption := GetDescrTabCol('CONTRIBUINTES_IMOB', 'NOME_FANTASIA');
);
```

Resultado:

Nome Fantasia / Mãe
---------------------

#### 8.1.1.10 GetDisplayLabelPlural

```
function GetDisplayLabelPlural(IdTabela: Integer; NomeTabela: WideString): String;
```

Retorna a descrição da tabela no plural que foi previamente cadastrada na aplicação **AMCTabela.exe**.

Não é necessário passar os dois parâmetros (IdTabela e NomeTabela), geralmente somente o NomeTabela é passado por parâmetro e IdTabela é passado como 0 (zero).

#### 8.1.1.11 GetDisplayLabelSingular

```
function GetDisplayLabelSingular(IdTabela: Integer; NomeTabela: WideString): String;
```

Retorna a descrição da tabela que foi previamente cadastrada na aplicação **AMCTabela.exe**.

Não é necessário passar os dois parâmetros (IdTabela e NomeTabela), geralmente somente o NomeTabela é passado por parâmetro e IdTabela é passado como 0 (zero).

#### 8.1.1.12 GetFormatDate

```
function GetFormatDate: String;
```

Retorna o formato de data padrão utilizado pelo SIM.

Formato de retorno: '**dd/mm/yyyy**'

#### 8.1.1.13 GetFormatTime

```
function GetFormatTime: String;
```

Retorna o formato de hora padrão utilizado pelo SIM.

Formato de retorno: '**hh:mm:ss**'

#### 8.1.1.14 GetInClause

```
function GetInClause(const FieldName: String; Ids: OleVariant): String;
```

Monta a clausula IN utilizadas em consultas SQL. **Ids** é um vetor com todos os valores que devem ser consultados.

Exemplo de utilização: Na variável **IdsContrib** possui 3 valores de IdsContribuintes, na posição 0, 1, 2 respectivamente:

IdsContrib = [1354, 7457, 6887]

Agora, deseja utilizar os valores que estão no vetor para fazer uma consulta SQL e retornar o nome dos contribuintes:

```
SELECT A.NOME_CONTRIBUINTE
      FROM CONTRIBUINTES A
     WHERE A.ID_CONTRIBUINTE IN (1354, 7457, 6887)
```

Utilizando a função:

```
SELECT A.NOME_CONTRIBUINTE
      FROM CONTRIBUINTES A
     WHERE + GetInClause('A.ID_CONTRIBUINTE', IdsContrib);
```

#### 8.1.1.15 GetLabelCaptions

```
function GetLabelCaptions(CaptionIds: OleVariant; out CaptionStrings,  
HintStrings: OleVariant): Boolean;
```

Retorna um vetor de rótulos e hints que foram previamente cadastrados na aplicação **AMCRotulo.exe**

##### Parâmetros de entrada:

*CaptionIds* : Colocar em cada posição do vetor, o IdRotulo que deseja-se buscar a descrição e o hint (ajuda).

##### Parâmetros de saída:

*CaptionStrings* : Vetor com a descrição dos rótulos.

*HintStrings* : Vetor com a descrição dos hints.

OBS: utilize esse método quando precisar configurar muitos rótulos que estão previamente cadastrados na aplicação, pois é muito mais eficiente do que chamar o método [GetRotulo](#) várias vezes. Nesse caso somente 1 chamada é feita ao banco de dados e todos os rótulos são retornados de uma só vez.

##### Exemplo de utilização:

```
procedure TfrCPDTool.ConfiguraCaptions;  
var CaptionStrings, HintStrings: OleVariant;  
begin  
  GetLabelCaptions(VarArrayOf([2092, 424, 775, 1568]), CaptionStrings, HintStrings);  
  TSPagamentos.Caption := CaptionStrings[0];  
  lbEntre1.Caption := CaptionStrings[1];  
  lbe1.Caption := CaptionStrings[2];  
  acbOrigem.Caption := CaptionStrings[3];  
end;
```

#### 8.1.1.16 GetMensagem

```
function GetMensagem(IdMsg: Integer): String;
```

Retorna a descrição da mensagem que foi previamente cadastrado na aplicação **GCAMMensagens.exe**

#### 8.1.1.17 GetRotulo

```
function GetRotulo(IdRotulo: Integer): String;
```

Retorna a descrição do rótulo que foi previamente cadastrado na aplicação **AMCRotulo.exe**

**8.1.1.18 GetServerDate**

```
function GetServerDate: TDateTime;
```

Retorna a data do servidor de aplicações.

**8.1.1.19 GetServerDateStr**

```
function GetServerDateStr: String;
```

Retorna a data do servidor de aplicações no formato padrão utilizado pelo SIM.

Formato de retorno: '**dd/mm/yyyy**'

**8.1.1.20 GetServerDateTime**

```
function GetServerDateTime: TDateTime;
```

Retorna a data e hora do servidor de aplicações.

**8.1.1.21 GetServerDateTimeStr**

```
function GetServerDateTimeStr: String;
```

Retorna a data e hora do servidor de aplicações no formato padrão utilizado pelo SIM.

Formato de retorno: '**dd/mm/yyyy' 'hh:nn:ss**'

**8.1.1.22 GetServerTime**

```
function GetServerTime: TDateTime;
```

Retorna a hora do servidor de aplicações.

**8.1.1.23 GetServerTimeStr**

```
function GetServerTimeStr: string;
```

Retorna a hora do servidor de aplicações no formato padrão utilizado pelo SIM.

Formato de retorno: '**hh:nn:ss**'

**8.1.1.24 GetTempDir**

```
function GetTempDir: String;
```

Retorna o caminho para o diretório de arquivos temporários do Windows.

**8.1.1.25 IsNull**

```
function IsNull(Value: Variant): Boolean;
```

Verifica se uma variável do tipo Variant está vazia.

### 8.1.1.26 MessageDlgCPD

```
function MessageDlgCPD(IdMsg: Integer; DlgType: TMsgDlgCPDType;
Buttons: TMsgDlgCPDButtons; HelpCtx: Longint; const HelpFileName: String;
Params: array of const): Integer;
```

Método que mostra uma caixa de diálogo no centro da tela com uma mensagem cadastrada no catálogo de mensagens.

#### Parâmetros de entrada:

*IdMsg* : ID da mensagem cadastrada no catálogo de mensagens.

*DlgType*: Define o tipo da mensagem a ser mostrada, adicionando à caixa de dialogo o ícone correspondente ao tipo escolhido.

Os tipos mais comuns utilizados são:

- mtcAviso;
- mtcErro;
- mtcInformacao;
- mtcConfirmacao;

*Buttons*: Define quais os botões que deverão aparecer na caixa de diálogo.

Os botões mais utilizados são:

- mbcSimNaoCancelar;
- mbcOKCancelar;
- mbcInformacao;
- mbcConfirmacao;

**OBS.** é possível informar o botão ou o conjunto de botões que deverão aparecer na caixa de diálogo. Veja no exemplo de utilização abaixo.

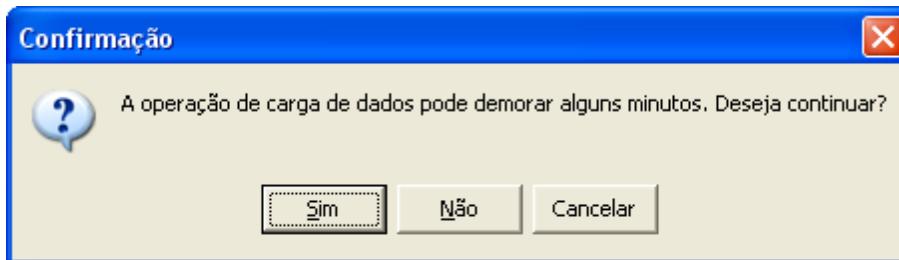
*HelpCtx*: Esse parâmetro apenas é utilizado quando um arquivo de help é associado com a aplicação. Serve como uma referência para o tópico correspondente no arquivo de help que pode ser acionado a partir da tecla F1. Caso não exista um tópico específico no arquivo de help ou até mesmo não exista tal arquivo, passar 0.

*HelpFileName*: informa o nome do arquivo de help associado à aplicação.

*Params*: Caso a mensagem contenha parâmetros, como por exemplo %s, %d..., é possível informá-los a partir deste array.

#### Exemplo de Utilização:

```
MessageDlgCPD(305, mtcConfirmacao, mbcSimNaoCancelar, 0, '', []);
```



```
MessageDlgCPD(2, mtcErro, [mbcOk], 0, '', [CliContribuinteImob.FieldName('NOME_CONTRIBUINTE').asString]);
```



#### 8.1.1.27 MessageDlgCPDStr

```
function MessageDlgCPDStr(Msg: String; DlgType: TMsgDlgCPDType;
Buttons: TMsgDlgCPDButtons; HelpCtx: Longint; const HelpFileName: String;
Params: array of const): Integer;
```

De maneira semelhante ao método MessageDlgCPD, mostra uma caixa de diálogo no centro da tela com uma mensagem digitada pelo desenvolvedor. Ao invés de utilizar uma mensagem cadastrada no catálogo de mensagens, apenas informar uma **string**.

#### 8.1.1.28 RemoteSGCA

```
function RemoteSGCA: ISGCADisp;
```

Retorna uma referência para a interface ISGCA. A partir dessa referência podem ser invocados os métodos que estão declarados em ISGCA.

### 8.1.1.29 RoundFloat

```
function RoundFloat (Value: Double; Digits: Integer): Double;
```

Realiza o arredondamento do valor **(Value)** passado, limitando-o ao número de dígitos informado.

### 8.1.1.30 SemErro

```
function SemErro(const Errors: CpErrors): Boolean;
```

Devolve *True* se **NÃO** existirem erros dentro da variável do tipo *ICpErrors* passada como parâmetro. Se existirem erros, devolve *False*.

### 8.1.1.31 StrToFloatDef

```
function StrToFloatDef(const S: String; Default: Extended): Extended;
```

Converte a variável *S* em Float, com o padrão passado em *Default*.

Parâmetros:

**S** - Variável a ser convertida.

**Default** - Padrão para a conversão. Caso ocorra algum erro durante a conversão, será retornado o valor **Default**

### 8.1.1.32 ValueInVarArray

```
function ValueInVarArray(Varia ntArray, Value: Variant): Boolean
```

Verifica se a variável **Value** existe dentro do vetor **Varia ntArray** passado como parâmetro.

### 8.1.1.33 VarArrayCount

```
function VarArrayCount(const A: Variant; Dimension: Integer): Integer;
```

Método que retorna a quantidade de valores em uma variável do tipo variant, ou seja, o tamanho do vetor.

Endnotes 2... (after index)

Back Cover