eticadata

ERP.18

PLATAFORMA & CUSTOMIZAÇÃO

Abr.2018 | DOC.MAN.45

CONTEÚDOS

INTRODUÇÃO

Este manual está dividido em duas grandes partes:

A primeira parte trata da plataforma genérica das aplicações ERP V18.

Fala de características que se aplicam a todos os módulos e que estão disponíveis de base, dependendo em alguns casos de módulos licenciados.

A segunda parte trata da extensibilidade da plataforma e dos mecanismos existentes para a mesma.

PLATAFORMA

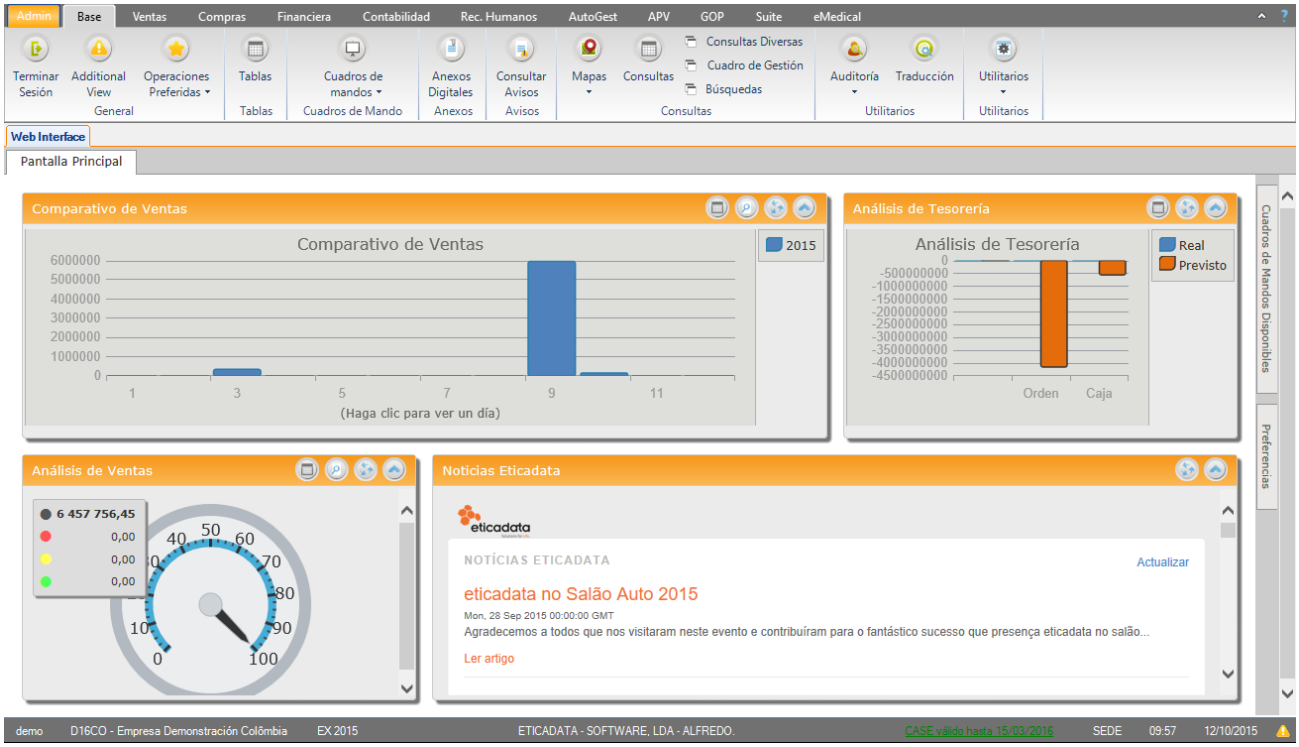
AMBIENTE DE TRABALHO

Os componentes do ERP V18 são feitos numa de duas tecnologias:

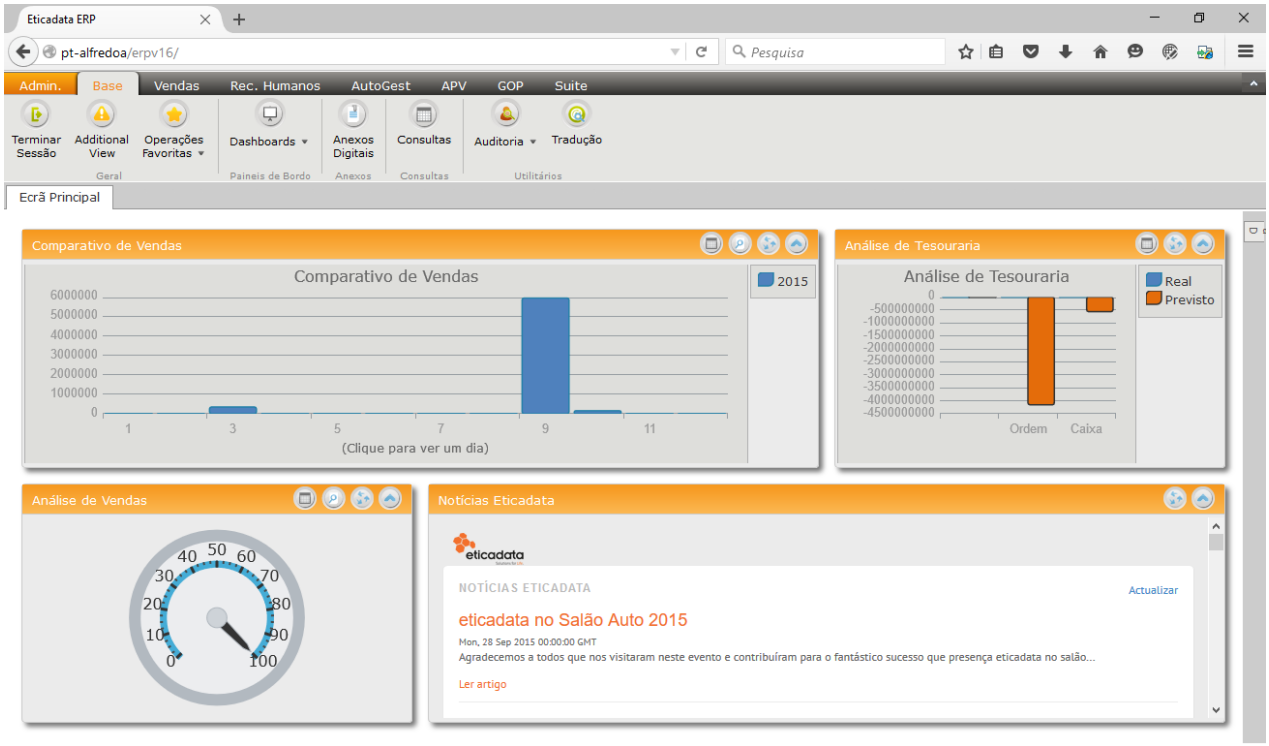
COMPONENTE DESKTOP ou COMPONENTE WEB.

As aplicações da Versão 18 do ERP eticadata (com exceção do POS) podem ser acedidas de duas formas:

- Uma aplicação *desktop* tradicional, iniciada por um atalho no ambiente de trabalho, no menu, ou diretamente pelo executável, que permite aceder a todos os componentes.



- Através de um *browser*, apontando para o site onde estão instalados os componentes web das aplicações, que só permite aceder a componentes web.



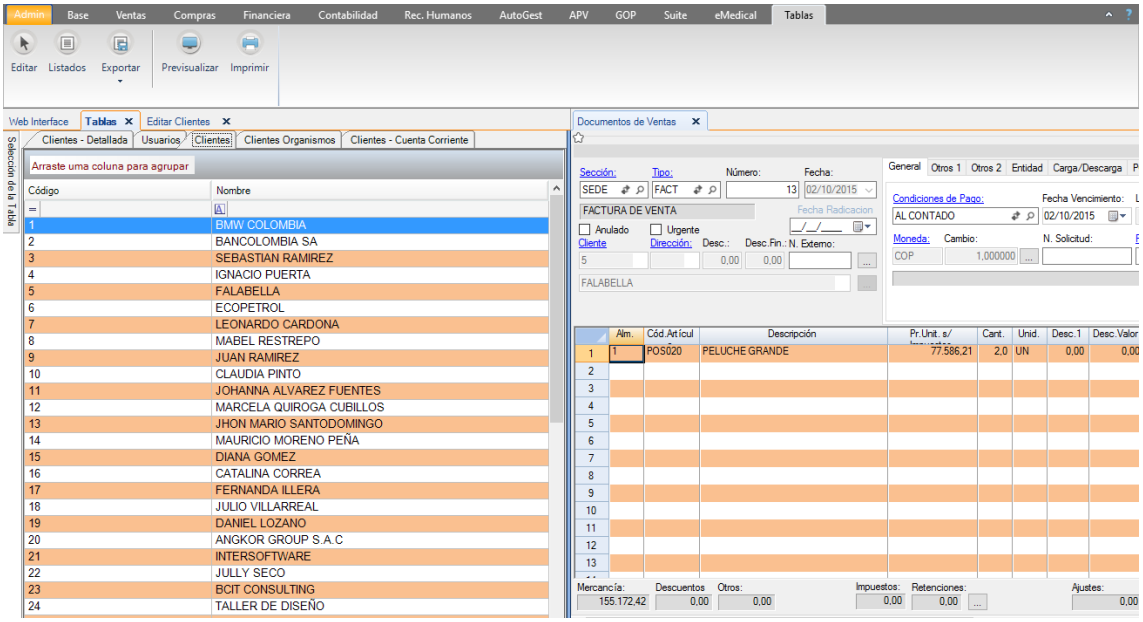
Apesar de haver duas formas de aceder aos componentes, ambos partilham as características básicas de navegação:

Têm uma *ribbon* no topo para navegar na aplicação, a partir da qual é possível invocar os vários componentes.

Esta *ribbon* tem uma área de administração (também chamada *backstage*) para tarefas pontuais de administração e preferências.

Tem uma área de trabalho onde são apresentados os componentes em separadores, que acrescentam uma aba contextual na *ribbon* para operações específicas.

Estes separadores podem ser arrastados e colados à esquerda, direita, acima ou abaixo, de modo a poder visualizar mais que um componente de cada vez.



Uma particularidade no *desktop* é que os componentes web não abrem separadores individuais, mas são apresentados num único separador chamado "Web Interface".

CONFIGURAÇÃO DA RIBBON

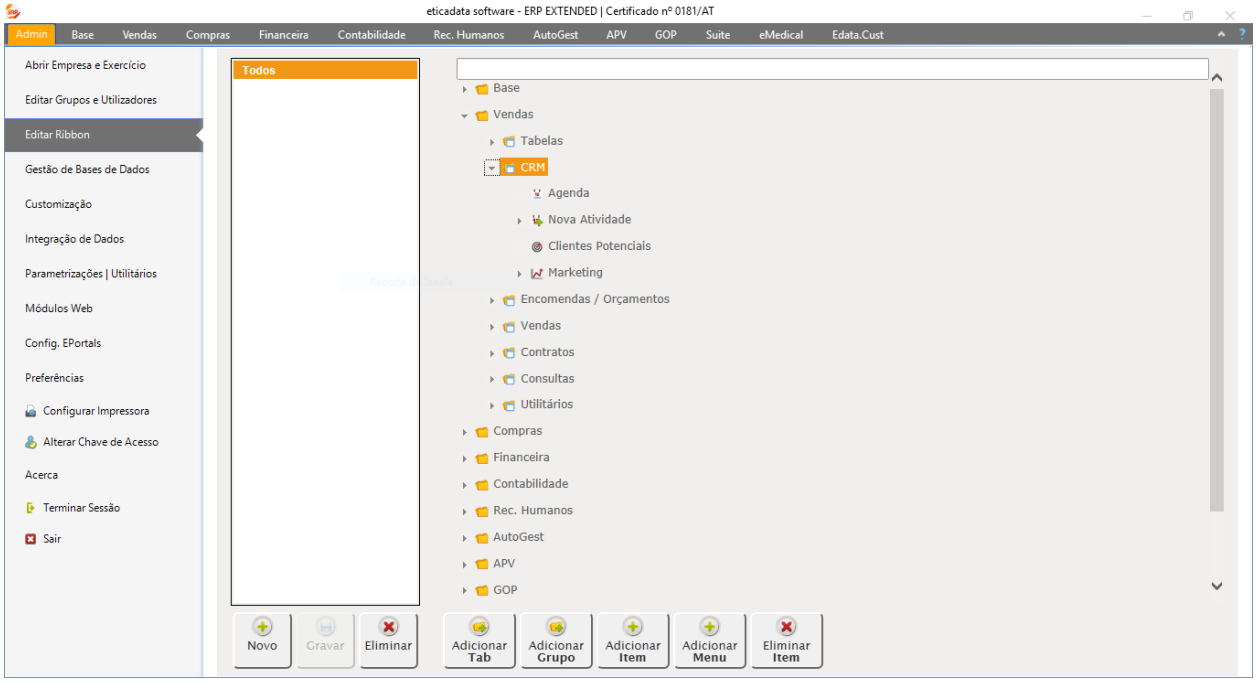
Conforme o licenciamento, mais especificamente a linha do produto, a *ribbon* pode se configurada, eventualmente com algumas limitações.

Assim, na linha BASIC, não é possível alterar a configuração da *ribbon*.

Na linha PREMIUM, é possível criar uma configuração da *ribbon* que se aplica a todos os utilizadores.

Na linha EXTENDED, é possível ainda criar configurações por grupo e por utilizador.

A edição das configurações da *ribbon* é feita através do menu de administração.



Na primeira caixa de seleção devemos selecionar a quem se aplica a configuração:

<Todos>, a um Grupo ou a um Utilizador.

Na segunda caixa de seleção selecionamos o grupo ou utilizador a quem se aplica.

Em seguida, clicamos em carregar para carregar a configuração aplicável à seleção que fizemos.

Caso não haja nenhuma configuração gravada, é carregada a configuração por defeito.

Se existir uma configuração, é possível removê-la utilizando o botão Eliminar, que ficará disponível nesse caso.

Depois de fazermos as alterações necessárias, deveremos carregar no botão Gravar para gravar as alterações.

As alterações efetuadas não serão visíveis na sessão atual.

Terá de iniciar uma nova sessão para as visualizar.

As alterações são feitas com recurso à árvore e aos botões à esquerda da mesma.

A árvore representa a hierarquia de separadores, grupos e itens na *ribbon*.

Podemos mover os elementos da *ribbon* arrastando-os.

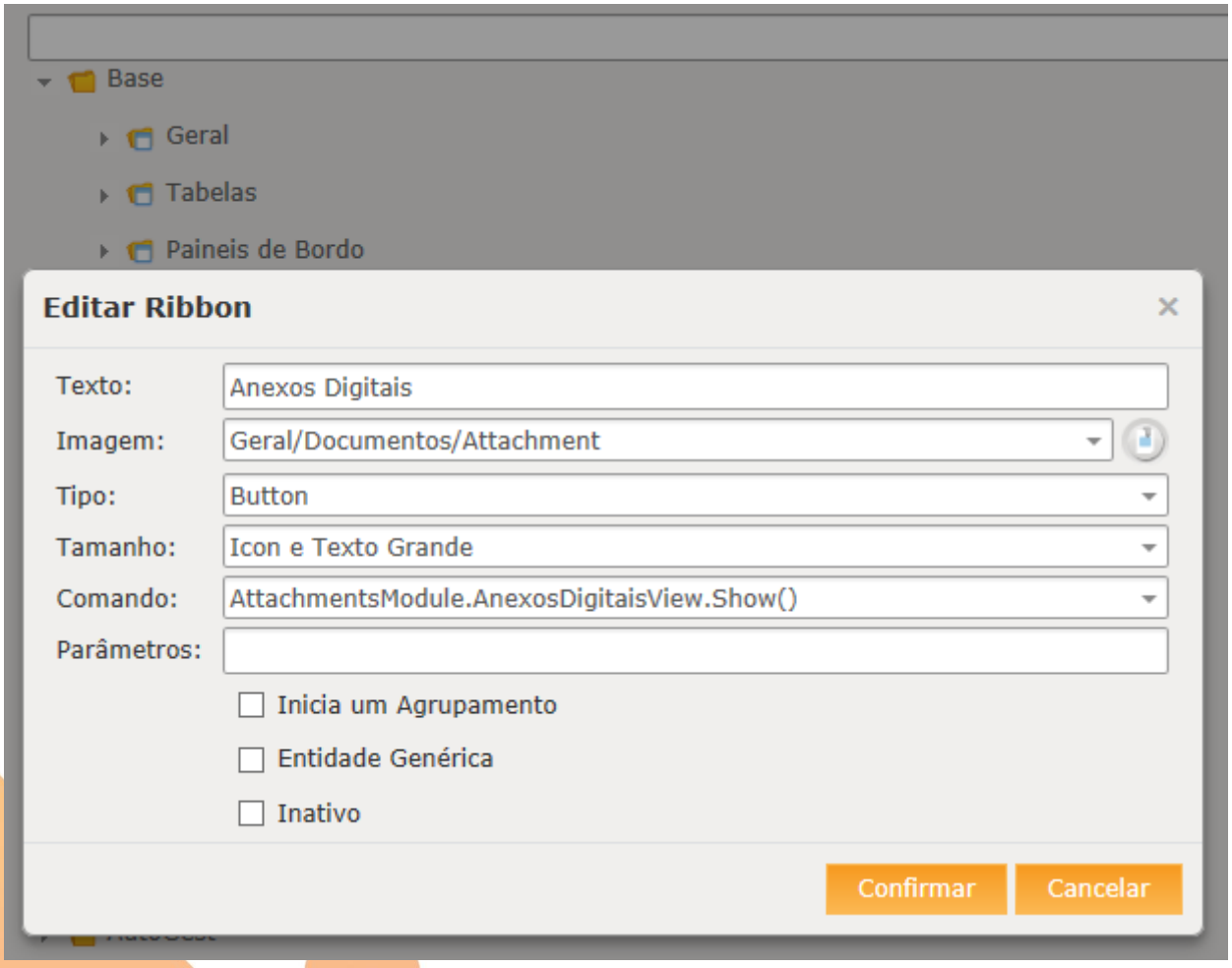
Para adicionar separadores, grupos ou itens, devemos usar os botões correspondentes.

Para eliminar qualquer tipo de item, devemos selecioná-lo e clicar no botão "Eliminar Item".

A edição dos itens faz-se dando dois cliques no item.

Para os separadores e grupos, a única alteração possível é no texto.

Para os restantes itens, podem-se alterar várias propriedades:



- O texto que é apresentado na *ribbon*.
- A imagem que é apresentada na *ribbon*.

  Esta pode ser inserida manualmente, ou arrastada do separador imagens, que apresenta todas as imagens que estão presentes na instalação servidor \imagens\icons.
- O tipo de objeto a apresentar na *ribbon*.
- O tamanho do objeto da *ribbon*.
- O comando a executar.

  Os comandos despoletam uma ação a que a aplicação vai responder.

  Tipicamente, provocam a abertura de janelas que permitem ao utilizador interagir com o sistema.

  Os comandos disponibilizados pela eticadata podem ser encontrados no separador Comandos.

  Os comandos que terminam com ".Show()" ou ".Execute()" são comandos que serão executados no contexto Web.

  Os comandos que começam por "#/" são executados também em contexto Web, sendo neste caso interfaces em HTML.

  Os restantes são executados no contexto *desktop*.

Os comandos podem ainda ser invocações para URLs ou aplicações externas.

Para um URL, deveremos inseri-lo na forma *http://<site>*.

Para as aplicações ou ficheiros externos, deverá ser na forma *file://<caminho para o ficheiro>*.

- Os "Parâmetros" dependem do comando que pode ou não aceitar parâmetros.

  Por exemplo, para apresentar uma consulta personalizada, podemos passar a identificação da consulta como parâmetro.
- "Inicia um agrupamento" indica que é pretendido um separador entre o item atual e o anterior.
- "Entidade genérica" indica que o item será excluído se não houver mais nenhum item ativo no seu separador que não seja entidade genérica.

Para executar um comando personalizado, terá de ser adicionado o módulo Silverlight ou o Assembly Desktop que contém esse comando.

Para adicionar o módulo Silverlight, temos de abrir a janela "Parametrização de Módulos Web", no menu Admin. Aí, clicar no botão "Adicionar Módulo" e selecionar um ficheiro .Xap que esteja na pasta ClientBin

do site Web de suporte ao ERP.

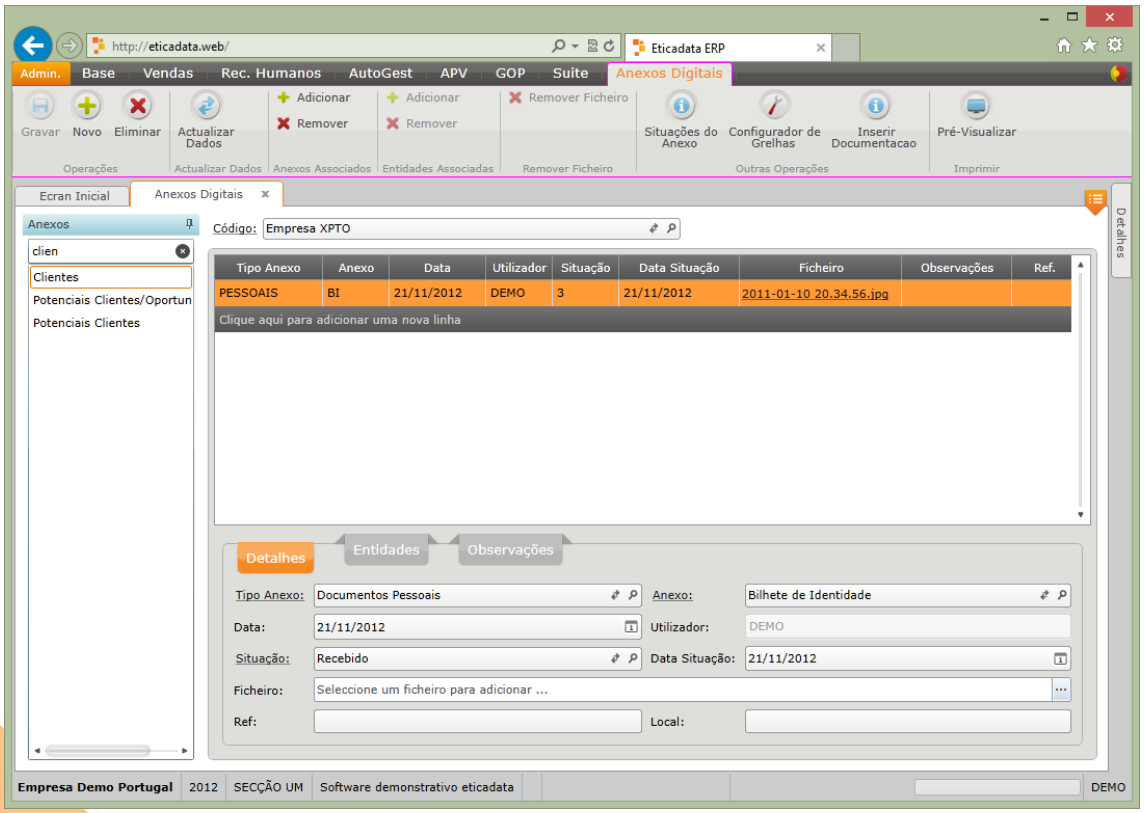
Para adicionar um Assembly Desktop, devemos ir a Customização \ Assemblies, também no menu Admin, e adicionar o Assembly desejado, indicando que contém comandos para serem executados pela Ribbon. Estes assemblies serão distribuídos para todos os postos da rede que necessitem deles, através da atualização automática de postos.

ANEXOS DIGITAIS

Os anexos digitais são uma característica que permite a associação de ficheiros (ou a emissão de relatórios) a registos do ERP.

Esta associação mantém um estado e o histórico desse estado.

Na V18, a gestão dos anexos digitais é feita através de um componente Web.



Para efetuar ou consultar um Anexo digital, primeiro temos de selecionar a entidade à qual este está associado. Para isso, selecionamos na árvore à esquerda o tipo de entidade, ou usamos a caixa de procura para mais facilmente encontrarmos o tipo de entidade pretendido.

Consoante o tipo de entidade, no topo da janela aparecem caixas de seleção que permitem localizar a entidade.

Uma vez localizada a entidade, são apresentados na grelha os anexos que estão associados a essa entidade.

Parar editar um registo, seleciona-se a linha correspondente.

Para remover um registo utiliza-se o botão "Remover anexos associados".

Para adicionar um registo, clicamos no botão "Adicionar anexos associados" ou na última linha da grelha.

Um anexo pode estar associado a mais que uma entidade.

Por exemplo, um contrato pode estar associado ao mesmo tempo a um cliente e a um documento de venda.

Para isso, no separador "entidades" podem-se editar as entidades adicionais às quais o anexo está associado.

Os anexos aparecem na grelha em ambas as entidades.

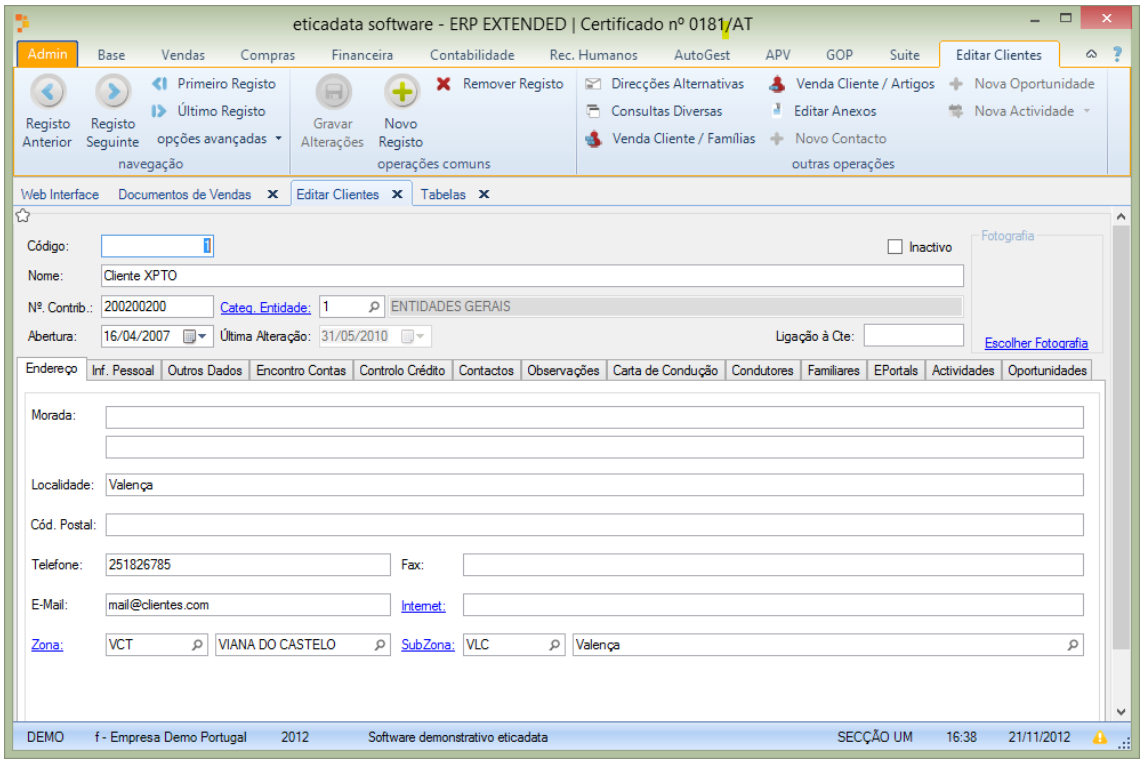
Os anexos ficam guardados no site de suporte ao ERP V18, numa pasta própria, numa pasta individual por anexo, com o nome original.

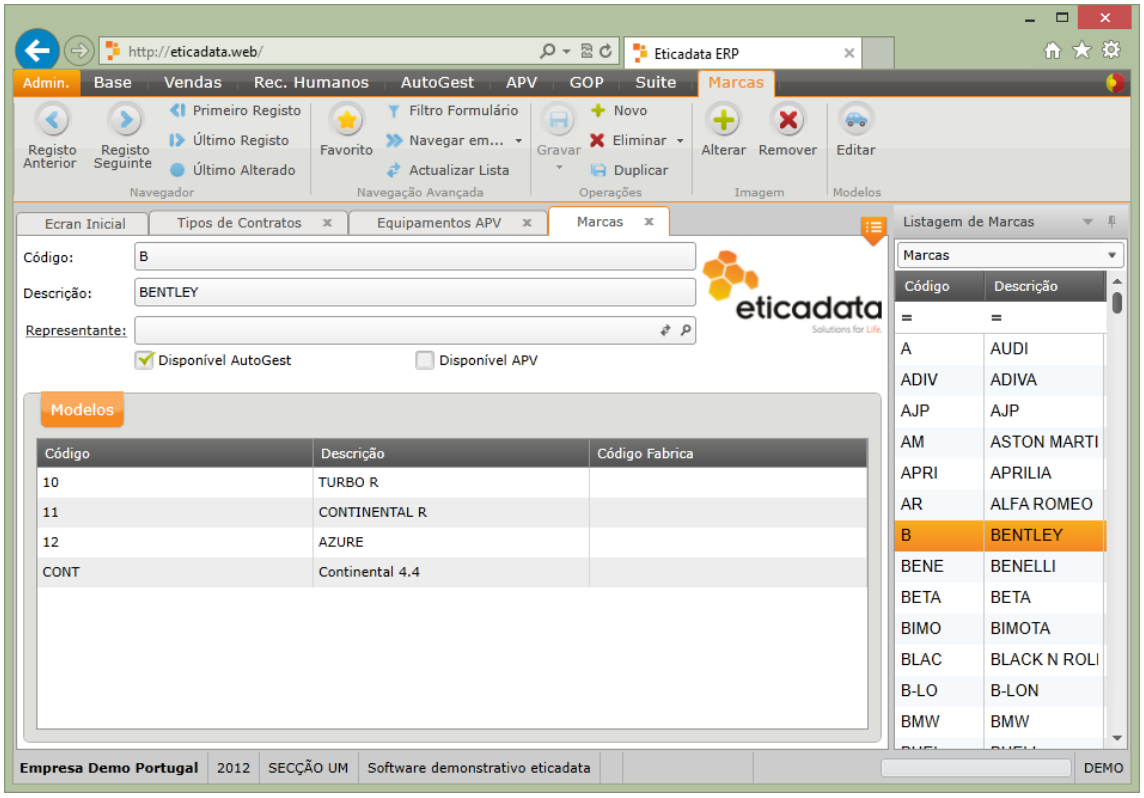
A migração de anexos da V9 para a V18 tem de ser feita por um utilitário próprio.

NAVEGAÇÃO AVANÇADA

As características de navegação avançada entre registos são diferentes para os componentes *desktop* e para os componentes web.

Neste capítulo apresentam-se as diferenças entre os dois ambientes.



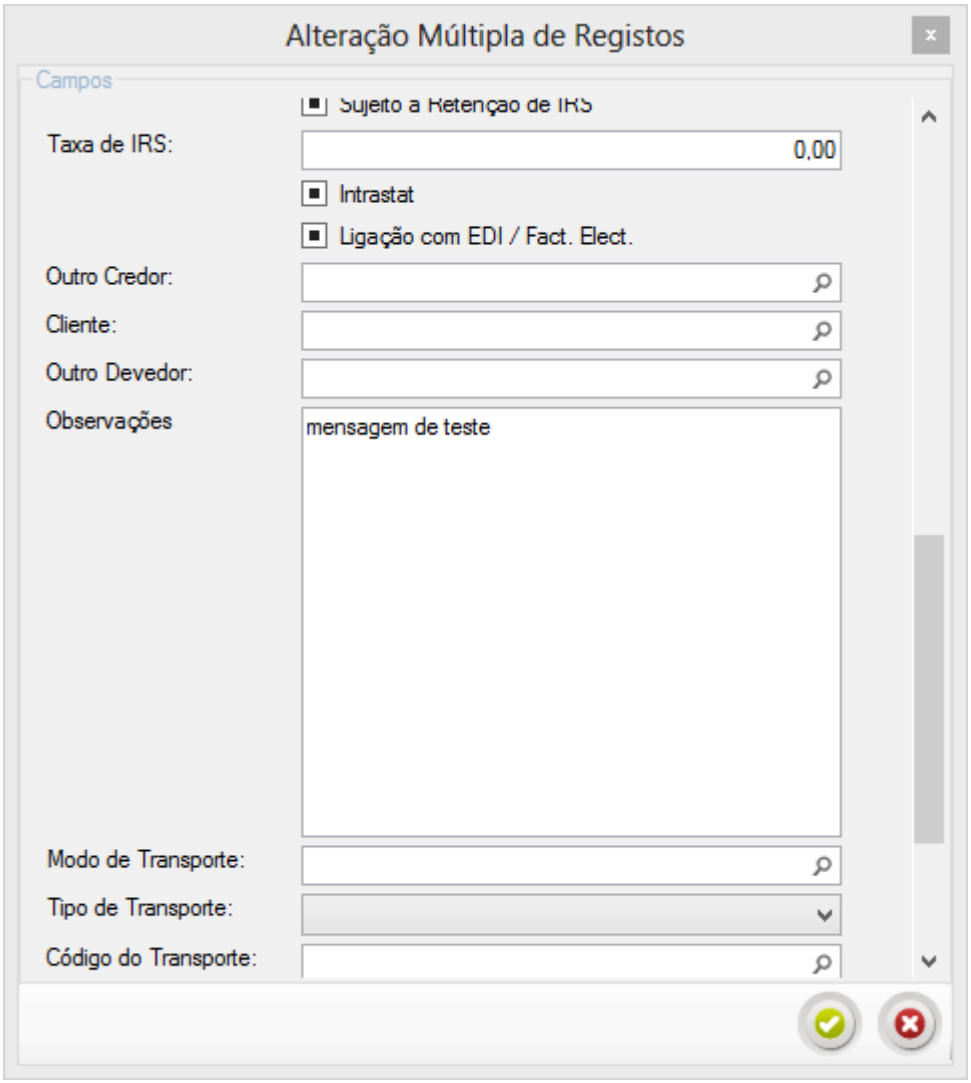

http://eticadata.web/
Eticadata ERP
Admin.
Base
Vendas
Rec. Humanos
AutoGest
APV
GOP
Suite
Marcas
Registo Anterior
Registo Seguinte
Primeiro Registo
Último Registo
Último Alterado
Navegador
Favorito
Filtro Formulário
Navegar em...
Actualizar Lista
Navegação Avançada
Gravar
Novo
Eliminar
Duplicar
Operações
Alterar
Remover
Imagem
Editar
Modelos
Ecran Inicial
Tipos de Contratos
Equipamentos APV
Marcas
Código: B
Descrição: BENTLEY
Representante:
Disponível AutoGest
Disponível APV
eticadata
Solutions for Life.
Modelos
Código
Descrição
Código Fabrica
10 TURBO R
11 CONTINENTAL R
12 AZURE
CONT Continental 4.4
Listagem de Marcas
Marcas
Código
Descrição
A AUDI
ADIV ADIVA
AJP AJP
AM ASTON MARTI
APRI APRILIA
AR ALFA ROMEO
B BENTLEY
BENE BENELLI
BETA BETA
BIMO BIMOTA
BLAC BLACK N ROLI
B-LO B-LON
BMW BMW
Empresa Demo Portugal
2012
SECÇÃO UM
Software demonstrativo eticadata
DEMO

	DESKTOP	WEB
MARCAÇÃO DE REGISTOS	No campo situado no canto superior esquerdo da janela	Na *ribbon*, botão "Favorito"
ÚLTIMO REGISTO ALTERADO	No menu opções avançadas	Botão "Último Alterado"
MARCAR / DESMARCAR TODOS OS REGISTOS DA LISTA	No menu opções avançadas	Não disponível
LISTA POR DEFEITO / LISTAS PERSONALIZADAS	No menu opções avançadas	No campo de seleção antes da lista, na área "Listagem de Marcas"
SÓ REGISTOS MARCADOS / DESMARCADOS	No menu opções avançadas	Navegar em "Apenas Favoritos", "Apenas Não Favoritos"
REGISTOS ALTERADOS APÓS A DATA	No menu opções avançadas	Navegar em "Alterados Após"
FILTRO POR CAMPOS PREENCHIDOS	No menu opções avançadas	Não disponível
DEFINIR FILTRO POR CAMPOS	No menu opções avançadas	Botão "Filtro Formulário"
REMOVER TODOS OS REGISTOS DA LISTA	No menu opções avançadas	Pop-Up do botão "Eliminar" – "Eliminar todos os registos da lista atual"
APRESENTAR LISTA ATUAL	No menu opções avançadas	Na área listagem (em ambiente silverlight)
ALTERAÇÃO MÚLTIPLA DE REGISTOS	No menu opções avançadas	Não disponível

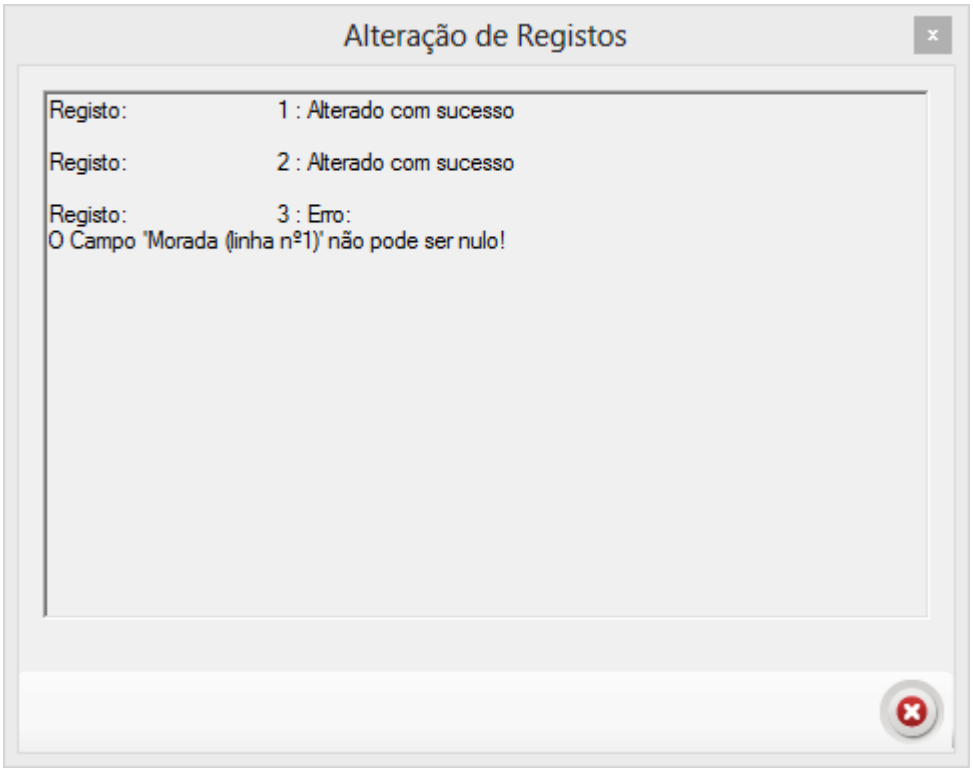
A alteração múltipla de registos consiste em efetuar a mesma alteração a todos os registos na lista atual de navegação.

Para a utilizar, devemos primeiro passar a usar uma lista que contenha todos os registos que pretendemos alterar e só esses.

Em seguida, ao clicar na opção, é apresentada uma janela com os campos passíveis de serem alterados, onde devemos introduzir a informação a colocar em todos os registos:


Alteração Múltipla de Registos
Campos
Sujeito a Retenção de IRS
Taxa de IRS:
0,00
Intrastat
Ligação com EDI / Fact. Elect.
Outro Credor:
Cliente:
Outro Devedor:
Observações
mensagem de teste
Modo de Transporte:
Tipo de Transporte:
Código do Transporte:

Ao confirmar, a aplicação irá aplicar a alteração a todos os registos e apresentará uma mensagem de resumo com as operações efetuadas e os erros encontrados:



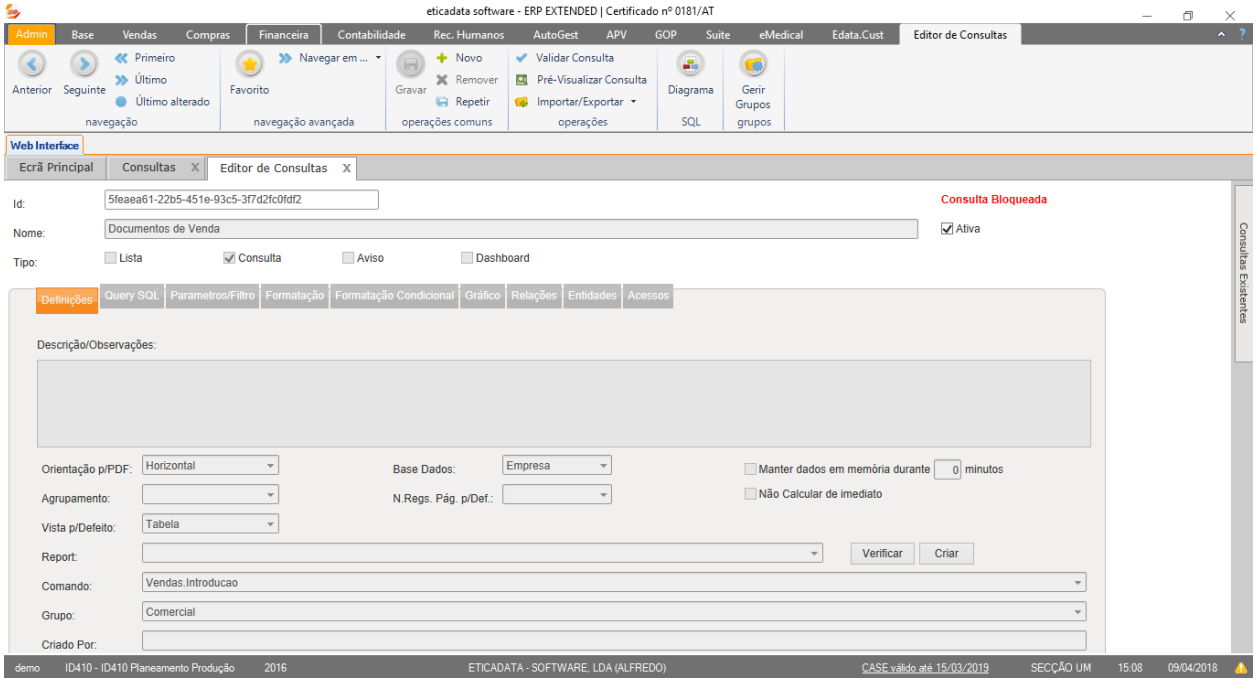
LISTAS E CONSULTAS PERSONALIZADAS

As consultas personalizadas são uma característica do ERP eticadata V18 que permitem a criação e personalização de listas de registos que podem depois ser utilizadas em várias circunstâncias.

A criação das listas e consultas personalizadas é um componente Web.

A sua consulta varia conforme o sítio onde seja visualizada.

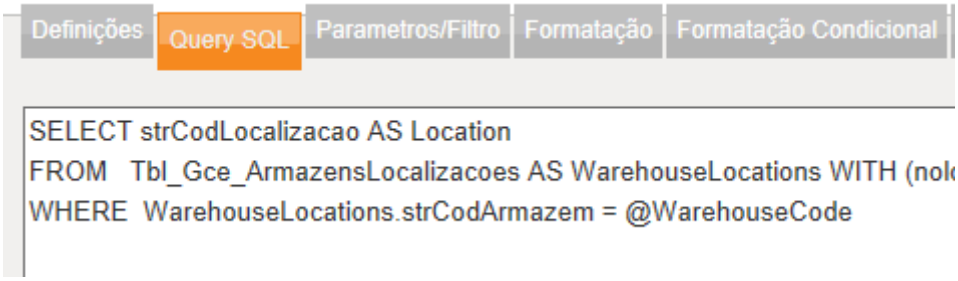
Para aceder à edição de consultas personalizadas, temos de ir à opção de consultas, no separador Base, posicionarmo-nos numa consulta, e clicar no botão "Editar Consulta".



Para criar uma consulta, podemos indicar os seguintes dados:

- Nome – Nome que será apresentado nas listas ou noutros locais para identificar a consulta.
- Comando – Quando for dado um duplo clique numa linha da consulta, será invocado o comando aqui indicado. Este comando irá despoletar uma ação no ERP, tipicamente apresentando uma janela.
- Tipo – Indica onde estará disponível a consulta:
    - Lista – A consulta será apresentada como uma lista adicional na janela de tabelas, para o comando indicado, e noutro sítios onde seja necessária uma lista para o comando indicado.
    - Consulta – A consulta estará disponível na janela de consultas.
    - Aviso – A consulta pode servir de base a avisos.
    - Dashboards – A consulta pode servir como fonte de dados para um *dashboard*.
- Ativa (On/Off) – A consulta está ou não ativa e aparece ou não nos sítios devidos.
- Definições
    - Observações – Texto livre que descreve a consulta.
    - Manter dados em memória - Permite indicar o intervalo em minutos para que os dados calculados se mantenham na memória, sem que sejam novamente calculados.
    - Nº Registos – Indica o número de registos pretendido, carregados da base de dados em cada bloco. Conforme o número de registos por bloco indicado, serão carregados da base de dados tantos registos quantos indicados por cada pedido à base de dados. Deverá ser usada uma definição adequada para evitar constrangimentos de performance.
    - Report – Permite indicar o nome do *report* a emitir quando se tentar emitir a consulta. Para criar um novo *report*, deve inserir um novo nome e clicar no botão "Criar Novo". Para atualizar a consulta num *report* já existente, deve clicar no botão "Verificar". Os *report*s gerados por esta opção são criados na raiz do site \reports\consultas. Devem ser editados e embelezados após a criação.
    - Pasta – Grupo onde o *report* será anexado.
    - Criado por – Indica o utilizador que criou a consulta.
    - ID – Identificação unívoca da consulta.

- Query SQL – *Query* que permite obter os dados da consulta. Esta *query* pode conter as variáveis que representam os parâmetros apresentados na área parâmetros:



- Parâmetros – As consultas podem não ser fixas e variar segundo alguns parâmetros. Aqui, podemos indicar esses parâmetros, que em alguns casos não estão disponíveis para ser usados (por exemplo se a consulta é usada como lista).
    - Nome – Nome do parâmetro, para ser usado na Query Sql.
    - Descrição – Descrição do parâmetro que é apresentado ao consultar, para o utilizador identificar os parâmetros.
    - Tipo de dados – Identificação da forma de introdução dos valores a serem passados para a consulta. O parâmetro terá tipos de dados diferentes, de acordo com o tipo indicado.
    - Tamanho / Casas decimais – Conforme o tipo de dados, limita e formata o valor a inserir.
    - Introdução – Indica como vai ser apresentado o campo ao utilizador.
        - Livre – O Utilizador pode introduzir o valor que pretender.
        - Com validação – O utilizador tem de selecionar um valor a partir de uma lista que lhe é apresentada.
        - Sem validação – É apresentada uma lista, mas o utilizador pode não escolher nenhum valor da lista e inserir um que não esteja na lista.
        - Não editável – O parâmetro é apresentado, mas o utilizador não pode alterar o seu conteúdo.
        - Oculto – O parâmetro é passado para a consulta, mas o utilizador não o vê.
    - Valor por defeito – Valor que o campo terá quando não for apresentado ao utilizador. Neste campo podemos usar as seguintes Macros:
        - %%SEC – Código da Secção ativa
        - %%USR – Login do utilizador atual
        - %%EX – Código do exercício atual.

- %%TODAY – Data atual, no servidor.
- %%STARTDATEOFMONTH – Primeiro dia do mês atual.
- %%ENDDATEOFMONTH – Último dia do mês atual.
- %%STARTDATEOFYEAR – Primeiro dia do ano atual.
- %%ENDDATEOFYEAR - Último dia do ano atual.
- %%CODARTIGOACTUAL – Código do artigo atual, caso a consulta tenha origem num movimento.
- %%CODARMAZEMACTUAL - Código do armazém atual, caso a consulta tenha origem num movimento.
- %%STARTDATEOFEX – Data inicial do exercício atual.
- %%ENDDATEOFEX – Data final do exercício atual.
- %%NOVOREGISTO – Indica se o registo atual é novo ou não, caso a consulta tenha origem num movimento.

- Valor de teste – Valor usado quando se clicar no botão da Ribbon "Testar Consulta".
- Lista
  - Tipo de dados – Indica se o campo vai ou não apresentar uma lista e qual o seu tipo.
    - Valores fixos que serão indicados no campo lista de dados, separados por ",".
    - Query Sql que será inserida no campo lista de dados.
    - Tabela que será indicada no campo tabela.
  - Valor visível – Caso seja usada uma Query Sql ou uma tabela, qual o campo que será apresentado ao utilizador, quando selecionar um elemento da lista.
  - Valor de filtro – Caso seja usada uma Query Sql ou uma tabela, qual o campo que será usado na query, quando for selecionado um elemento da lista.
  - Tamanho das colunas – Quando for usada uma Query Sql, qual o tamanho pretendido para as colunas, separadas por ",".

- Formatação – Esta secção permite formatar as colunas das consultas. De notar que algumas das formatações poderão estar apenas disponíveis quando a consulta é apresentada na Web. Após a revisão da Query Sql, deve testar a consulta para refrescar a informação de formatação inserindo / removendo as colunas necessárias.
  - Cabeçalho – Texto que identifica a coluna.

- Chave – Indica que a coluna representa um dos campos chave que é passado para o comando invocado. Caso a coluna seja chave, deve indicar o campo a que corresponde.
- Visível – Indica se a coluna estará visível ou não para o utilizador.
- Grupo e Ordem (Grupo) – Indica se a coluna estará agrupada por defeito ou não, e qual a sua ordem, caso haja mais do que uma coluna agrupada.
- Totais – Indica se a coluna vai ou não permitir apresentar totais.

- Formatação condicional – Aqui, podemos definir condições que, uma vez verificadas, irão alterar o formato de uma linha ou de uma célula ao nível da cor ou do tipo de letra, permitindo-nos salientar registos. A condição pode comparar dois valores de uma mesma linha, ou comparar o valor de uma célula com um valor fixo. As condições podem ser complementares, ou seja, só são avaliadas se uma das anteriores não teve sucesso.
- Gráfico – Aqui, podemos indicar se a consulta poderá ser vista como gráfico ou não, e, caso seja, quais as suas características.
- Relações – Caso queiramos relacionar duas consultas, para que uma delas apareça como detalhes da outra, deveremos inserir uma linha e indicar a coluna da nossa consulta que irá ser relacionada com outra consulta e com que coluna dessa consulta.
- Permissões – Aqui, podemos indicar os utilizadores e as empresas às quais a consulta se aplica. Esta só estará disponível nos vários pontos da aplicação caso estejamos numa empresa válida e num utilizador ou grupo válido. De notar que as definições são todas por inclusão, ou seja, se um utilizador pertence a um grupo e o grupo tem permissão, então o utilizador irá ter acesso à consulta, mesmo que não tenha sido dado acesso explicitamente a esse utilizador.

Na opção de consultas, além de podermos ver a consulta em formato de tabela, podemos ainda:

- Exportar para Excel ou Csv.
- Ver a consulta em formato de Pivot.
- Ver a consulta em formato de Report e imprimi-la ou guardá-la em PDF.
- Ver a consulta como gráfico e guardar uma imagem com a sua representação.
- Guardar vários conjuntos de valores com parâmetros predefinidos para uma posterior consulta rápida.

AVISOS PERSONALIZADOS

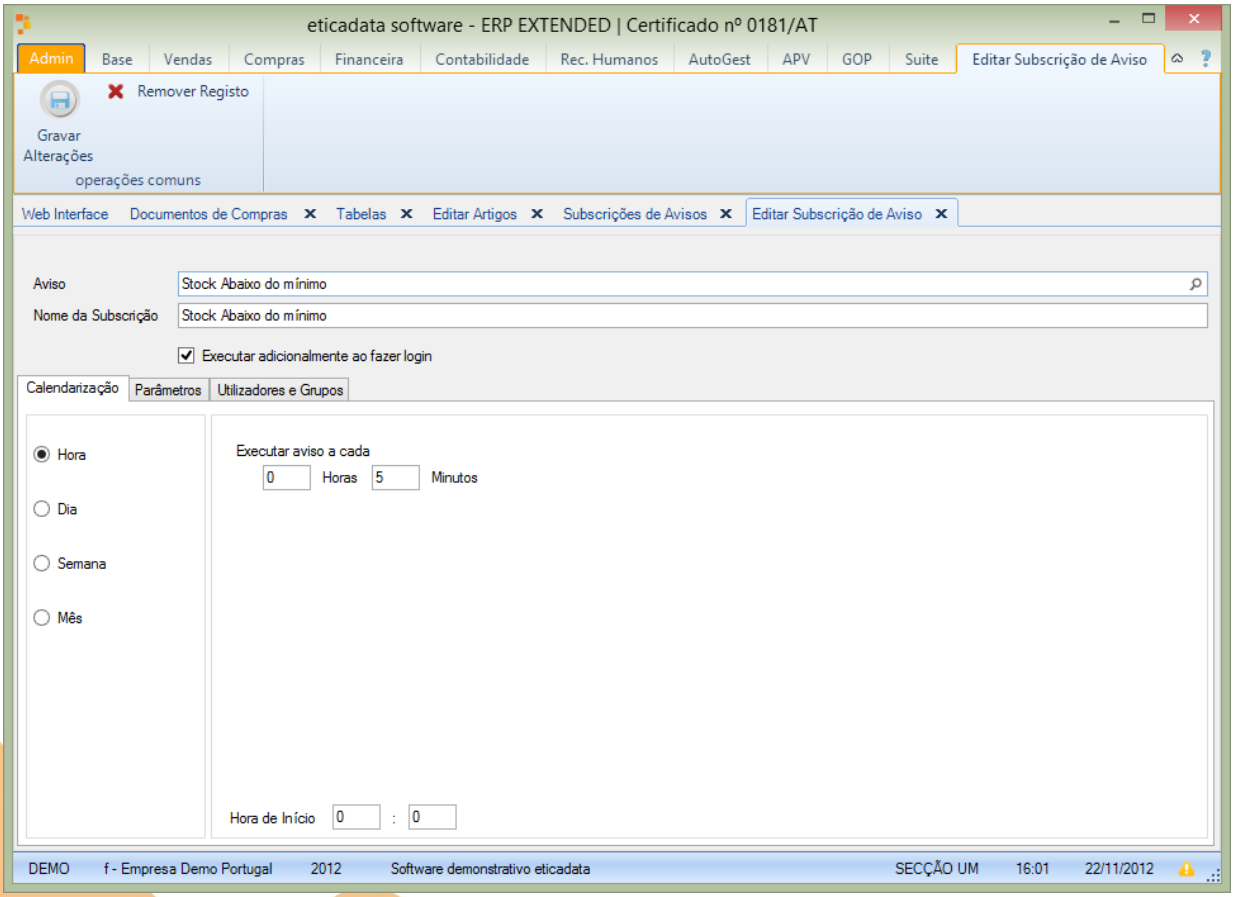
Os avisos personalizados baseiam-se em consultas.

A premissa para um utilizador ser avisado é que uma determinada consulta devolve registos.

Para usar uma consulta como base para um aviso, temos de a marcar como "Aviso".

Em seguida, no *desktop*, temos de clicar em "Subscrever avisos" para passar ao passo seguinte.

Nesse passo, vemos uma lista das subscrições existentes e, clicando no botão, podemos criar ou editar subscrições.



Na janela seguinte, podemos indicar qual a consulta a ser avaliada, quando queremos que o aviso seja despoletado, atribuir os parâmetros à consulta e indicar quem deve ser notificado.

A calendarização poderá ser:

- No início da aplicação, ao fazer Login.
- A uma determinada hora, repetindo de x em x minutos.
- A uma determinada hora, em vários dias, semanas ou meses.

Os parâmetros do aviso são herdados da consulta que pretendemos obter.

Podemos ter avisos mais regulares com parâmetros mais restritos, por exemplo, e consultas menos regulares com parâmetros menos restritos.

Por fim, podemos dizer a quem se aplica a consulta.

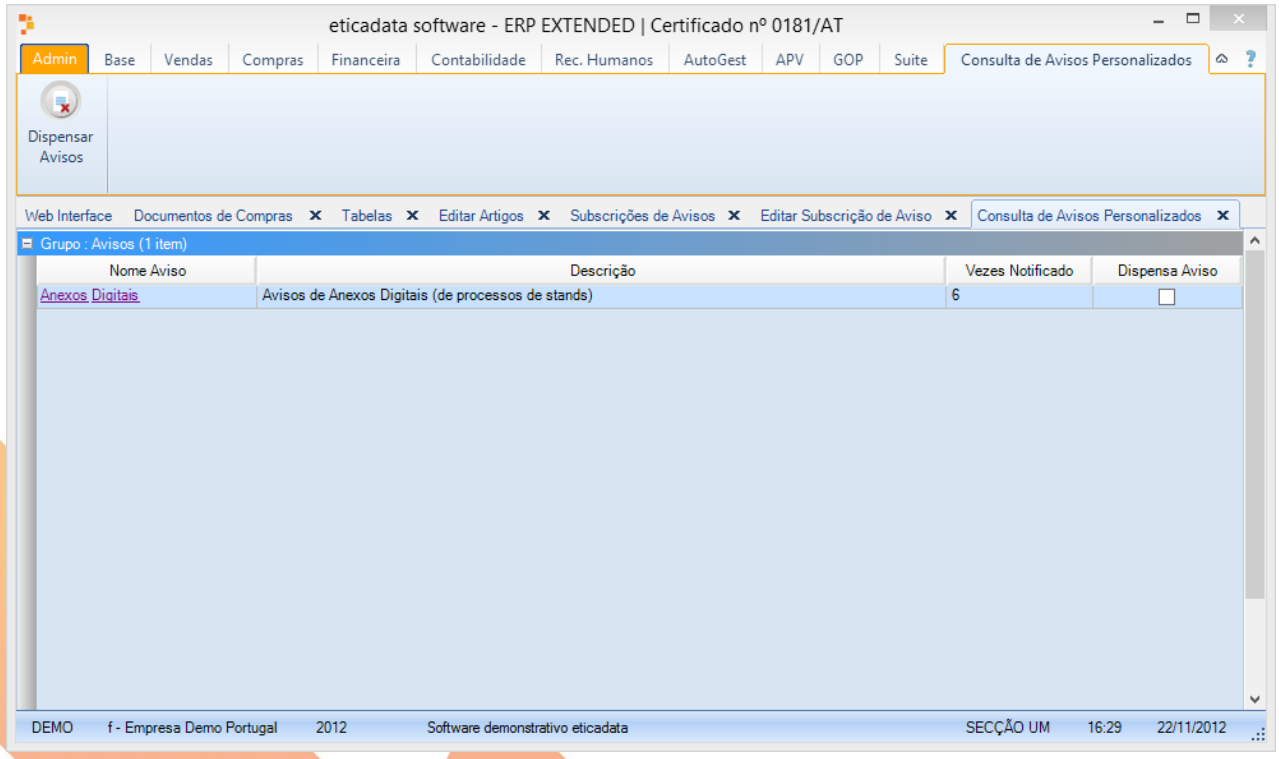
Os utilizadores indicados, quando entrarem numa determinada empresa ficarão com os avisos ativos.

Na hora da avaliação da consulta, se esta devolver registos, o utilizador será notificado.

A notificação surge no canto interior direito do ecrã, e desaparece após uns segundos.

A janela principal da aplicação fica com um símbolo no canto interior direito, indicando que há notificações que ainda não foram vistas.

O utilizador pode clicar nesse símbolo ou no botão "Consultar Avisos", na *ribbon* base para obter uma listagem de todos os avisos ainda não consultados.

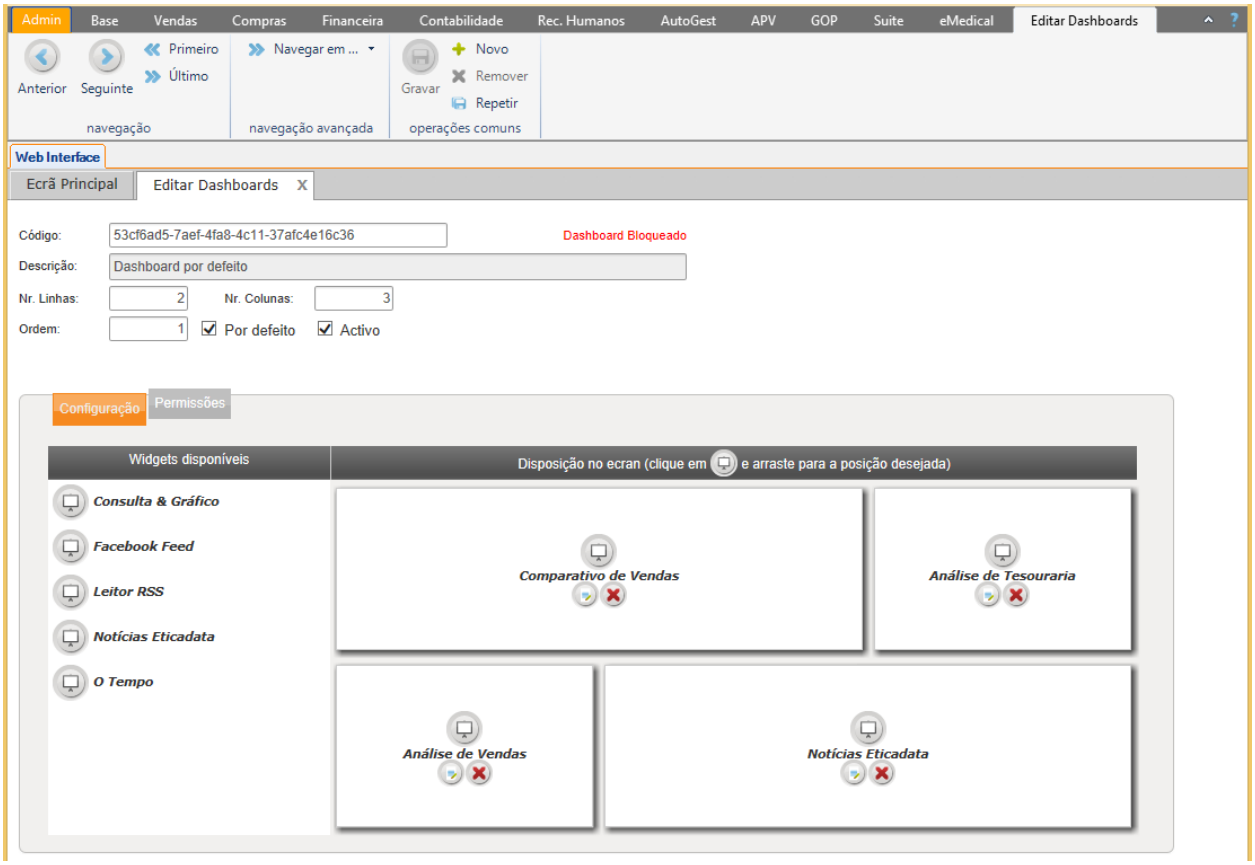


Nesta janela, o utilizador pode consultar os avisos, clicando no link de cada um. Quando já não necessitar da notificação, pode dispensar o aviso, clicando na coluna "Dispensa Aviso", na linha em causa, ou carregando no botão "Dispensar Avisos".

DASHBOARDS (PAINÉIS DE BORDO)

A funcionalidade de *dashboards* permite a criação de quadros resumo com a informação que o utilizador considera essencial para acompanhar a sua empresa. Os *dashboards* e a sua construção são componentes Web.

Para editar ou criar os *dashboards*, devemos clicar na dupla seta da galeria de Painéis de bordo e em seguida clicar em "Editor de dashboards".

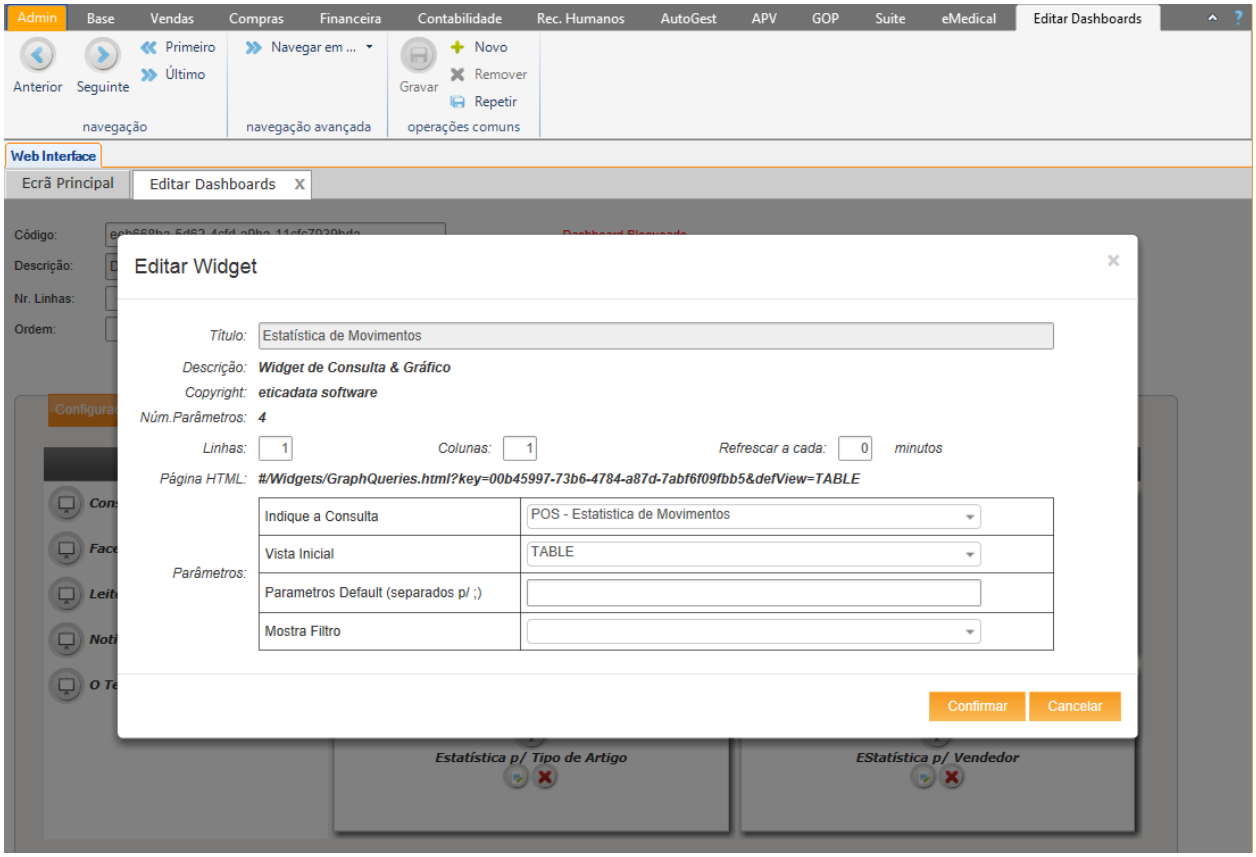


Os *dashboards* são compostos de *widgets*, que são apresentados numa grelha.

Temos de indicar a dimensão da grelha, consoante o tipo de informação a apresentar.

A dimensão do *widget* pode variar em função da informação que se pretende apresentar.

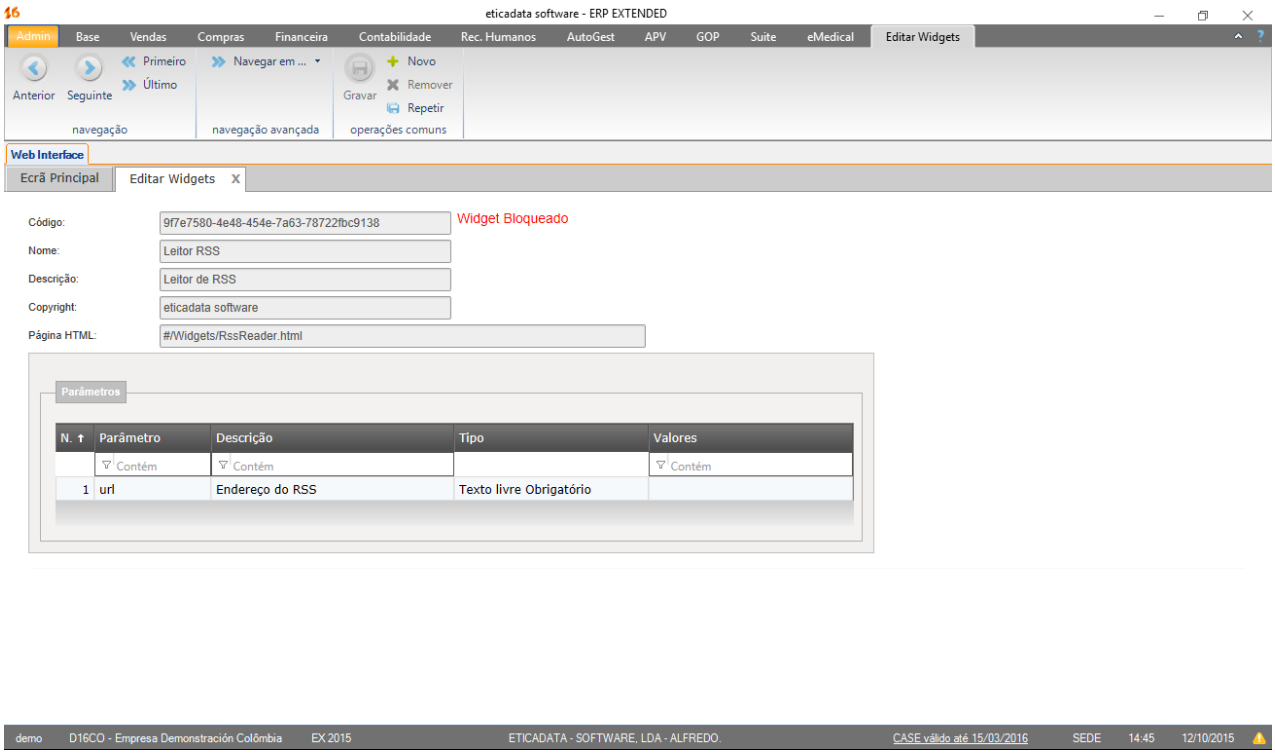
Na edição do *widget*, pode ser definida a quantidade de colunas e linhas que este irá ocupar.



Os *widgets* são páginas HTML, e podem ser acrescentados novos através da edição de *widgets*.

Na página de edição, deve indicar a localização da página HTML que contém a parametrização de interface do *widget*.

O endereço da página deve começar com #/Widgets/<NomePagina>.html, sendo que o ficheiro HTML deve estar disponível na pasta <DRIVE>\eticadata Sites\ERP V18\Eticadata.Web\Widgets\.



Para usar um *widget* num módulo, temos de o arrastar da área de *widgets* disponíveis para a lista de *widgets* do *dashboard* em edição.

Os *widgets* podem necessitar de parâmetros para obter dados ou para a sua configuração.

Nesse caso, ao selecionarmos o *widget* na linha, aparecem os dados a introduzir para a sua correta parametrização.

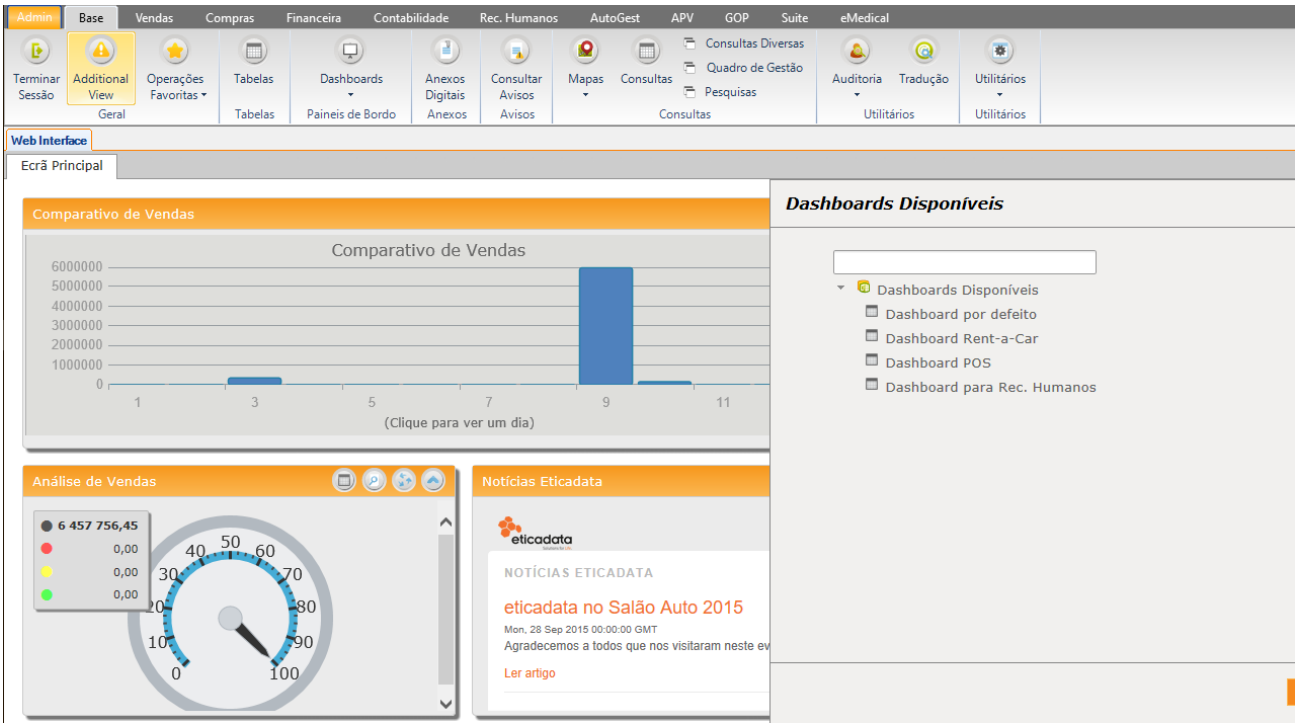
Existe ainda um parâmetro comum a todos os *widgets*.

Trata-se do parâmetro 'Atualizar a cada (minutos)', que permite indicar o intervalo de atualização automática dos valores do *widget*.

Podemos ainda indicar se o *dashboard* é um favorito. Nesse caso, ao iniciar a sessão, este será automaticamente aberto.

Podemos também indicar para que utilizadores e empresas o *widget* estará disponível.

Depois de inserir / editar os *widgets*, temos de reiniciar a sessão para que estes apareçam na *ribbon*.



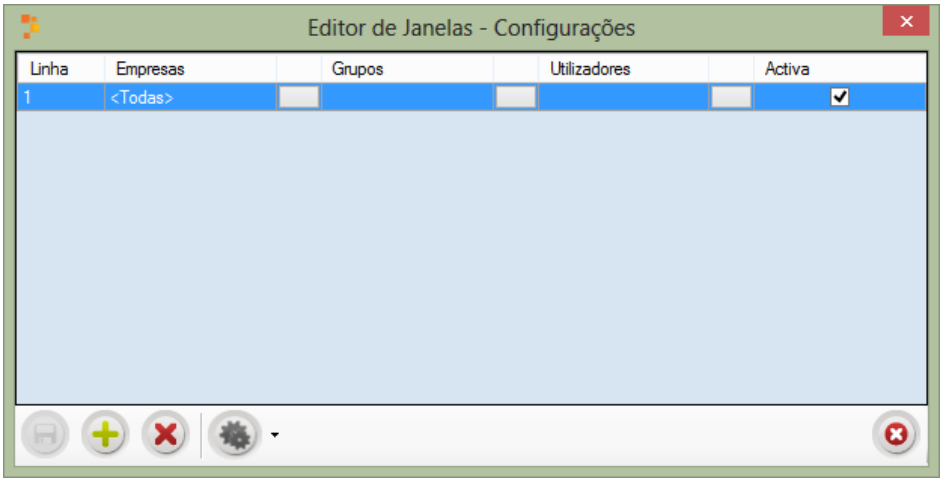
Ao clicar no *dashboard* pretendido, este é aberto e apresenta os *widgets* configurados.

Ao visualizar um *dashboard*, o utilizador pode usar os pequenos botões de redimensionamento para ver a informação com mais detalhe, podendo ainda aceder a mais funcionalidades, conforme o *widget*.

EDIÇÃO DE JANELAS

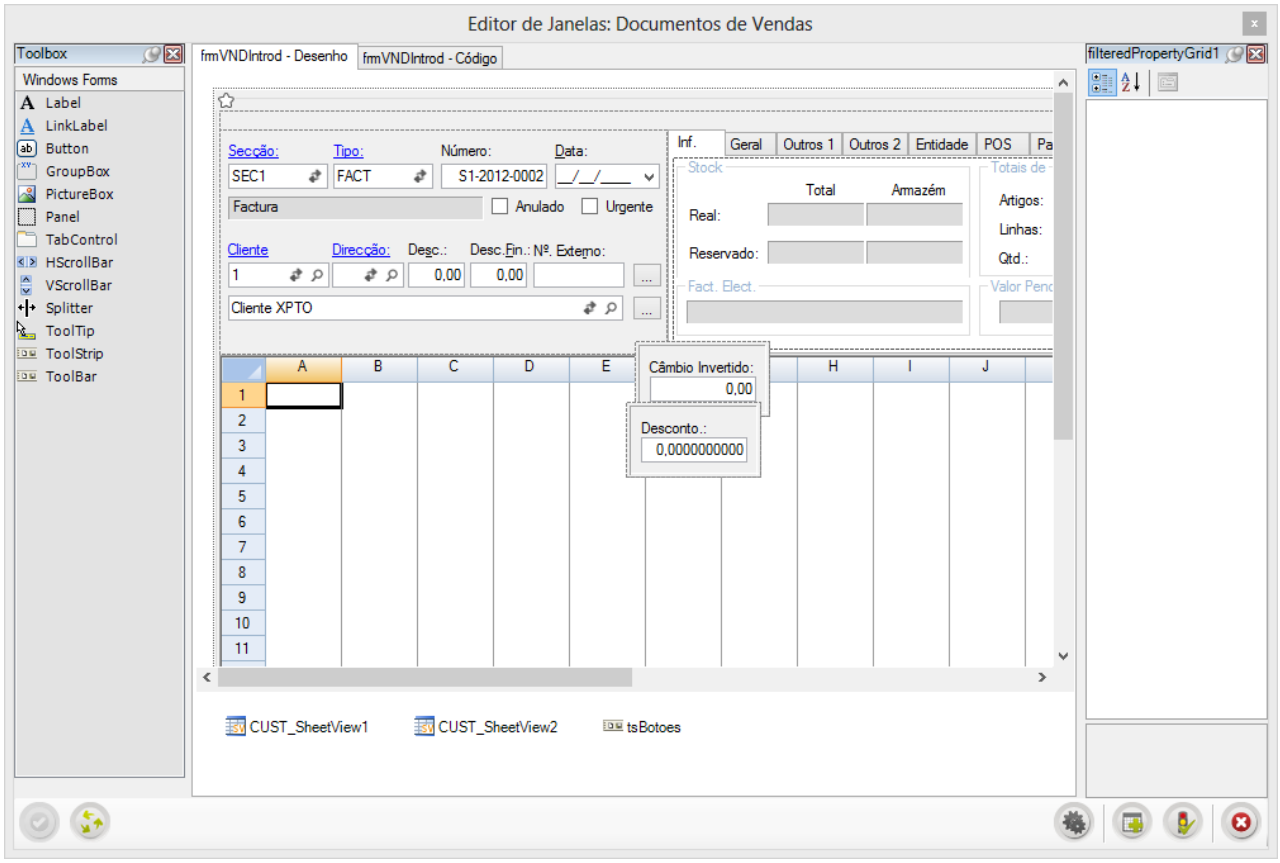
A edição de janelas é uma característica que permite adaptar o desenho das principais janelas da aplicação às necessidades dos clientes, normalmente simplificando o seu *design* ou dando mais destaque aos campos que os utilizadores mais utilizam.

A edição de janelas só está disponível para componentes *desktop*.
Na linha Premium, só é possível criar janelas editadas para todos os utilizadores ou grupos.
Na linha Extended, é possível criar janelas editadas por empresa, grupo e utilizador.
Para editar uma janela, devemos abri-la e selecionar a opção "Editar Janela" no menu "opções avançadas".



Nesta janela, podemos criar ou alterar várias configurações de janelas para vários cenários.
Se mais do que uma configuração se aplicar ao utilizador atual, será usada a 1ª configuração.
Caso não seja encontrada nenhuma configuração personalizada e ativa para o utilizador atual, na empresa atual, será usado o desenho da janela por defeito.

Para alterar o desenho de uma janela, devemos clicar no botão "Editar Janela" e, depois de fechar a janela editada, devemos gravar.
A janela de edição propriamente dita é composta por três áreas:



À esquerda, temos uma barra de ferramentas que podemos inserir na janela.

Ao centro, temos o desenho da janela, onde podemos selecionar os objetos que pretendemos alterar e redimensioná-los ou movê-los.
À direita, temos um painel que apresenta as propriedades do objeto selecionado, e onde podemos alterar os valores pretendidos.

Também é possível criar algum código para personalizar o comportamento da janela, embora, o seu uso seja limitado.
Não são aconselhados grandes desenvolvimentos com esta ferramenta.

Por defeito, as janelas disponibilizam os seguintes métodos que se podem programar:

```
Protected Overrides Sub PrepararJanela()
  MyBase.PrepararJanela()
End Sub

Protected Overrides Sub Antes_Posicionar()
  MyBase.Antes_Posicionar()
End Sub

Protected Overrides Function Antes_PreVisualizar(nomeFicheiro As String) As Boolean
  Return MyBase.Antes_PreVisualizar(nomeFicheiro)
End Function

Protected Overrides Function Antes_Imprimir(nomeFicheiro As String) As Boolean
  Return MyBase.Antes_Imprimir(nomeFicheiro)
End Function

Protected Overrides Function Antes_Gravar() As Boolean
  Return MyBase.Antes_Gravar()
End Function

Protected Overrides Function Antes_Remover() As Boolean
  Return MyBase.Antes_Remover()
End Function

Protected Overrides Function Antes_Fechar() As Boolean
  Return MyBase.Antes_Fechar()
End Function
```

```
Protected Overrides Sub Depois_Posicionar()
  MyBase.Depois_Posicionar()
End Sub

Protected Overrides Sub Depois_PreVisualizar()
  MyBase.Depois_PreVisualizar()
End Sub

Protected Overrides Sub Depois_Imprimir()
  MyBase.Depois_Imprimir()
End Sub

Protected Overrides Sub Depois_Gravar()
  MyBase.Depois_Gravar()
End Sub

Protected Overrides Sub Depois_Remover()
  MyBase.Depois_Remover()
End Sub

Protected Overrides Sub Depois_Fechar()
  MyBase.Depois_Fechar()
End Sub
```

O método PreparaJanela é executado depois do *design* da janela ser preparado.

O método AntesPosisionar ocorre antes de ser preenchida a janela com o registo corrente.

Os restantes métodos "Antes" ocorrem quando o utilizador clica nos botões correspondentes, se a operação existir na janela em causa, e os "Depois" ocorrem depois de todas as operações relativas à ação serem executadas.

Nos métodos "Antes", é possível devolver um valor (*cancel*) que permite à aplicação prosseguir ou não com a ação.

Após editar o código, temos que usar o botão "compilar" para verificar se o código que escrevemos está correto.

Após fecharmos a janela de edição e a janela de seleção de desenhos da janela, a aplicação vai fechar a janela em causa e podemos voltar a abri-la imediatamente para ver as alterações efetuadas.

Caso haja algum erro ao abrir a janela, a aplicação irá informar-nos do erro e abrirá o desenho de janela por defeito para podermos retificar o desenho em erro.

API DE INTEGRAÇÃO

A API disponibilizada pela eticadata permite efetuar todas as operações feitas pelo ERP. De facto, a API é a mesma usada pelo ERP.

A API distingue-se também entre componentes Desktop e Componentes Web.

Os componentes Desktop são acedidos por referência a Dlls disponíveis para o efeito. Os componentes Web são consumidos como WebServices.

INTEGRAÇÃO COM COMPONENTES DESKTOP

A eticadata disponibiliza as suas Dlls feitas em Visual Basic / C# .Net, que são desenvolvidas com o Visual Studio 2013 para a .Net FrameWork 4.5.1. Estas Dlls podem ser usadas em qualquer ambiente/linguagem de programação que suporte a referência direta das mesmas.

A) REFERÊNCIAS NECESSÁRIAS

Dada a extensão de entidades geridas pelo ERP eticadata, a gestão destas entidades é feita por um conjunto de Dlls em que cada uma é responsável por um conjunto de funcionalidades.

As Dlls são:

Eticadata.EtiVersions.dll – Contém constantes e enumerados necessários às operações.

Eticadata.ERP.dll - Contém o "Pai" de todas as classes, a partir do qual, uma vez instanciado e inicializado, se podem obter referências a todos os objetos do sistema.

Eticadata.Generics.dll – Contém as classes de acesso a dados, empresas, utilizadores, exercícios, campos adicionais, eventos, regras de validação, etc.

Eticadata.Tables.dll – Contém as classes de gestão das principais tabelas da aplicação.

Eticadata.Movs.dll – Contém as classes de gestão dos movimentos da Comercial, POS e Contabilidade.

Distribuição / versões das referências

A versão das Dlls tem de estar de acordo com a versão das bases de dados utilizadas.

Para garantir isso, deverá ser evitada a distribuição das Dlls da eticadata, pois não há garantia da versão que o utilizador tem.

Como a eticadata não coloca as suas Dlls na GAC, será necessário que as aplicações desenvolvidas sejam distribuídas para a mesma diretoria que as Dlls ou para uma diretoria anterior, sendo possível configurar o app.config indicando uma subdiretoria onde procurar as Dlls (ver os .config dos executáveis da eticadata).

Necessidade de instalar a aplicação

Naturalmente que estas Dlls fazem referência a mais algumas (além das Dlls da própria .Net Framework) para complemento das suas funcionalidades. Assim, caso seja mesmo necessário utilizar estas Dlls numa outra diretoria que não a da instalação da aplicação, é necessário copiar toda a diretoria "Bin" onde elas se encontram.

As Dlls que a eticadata produz são independentes do processador. No entanto, são utilizadas algumas Dlls de terceiros que apenas funcionam em processadores e sistemas operativos de 32 bits. Para garantir o correto funcionamento das aplicações em qualquer sistema, é aconselhada a compilação de qualquer executável com o atributo de compilação x86.

Licenciamento

Para executáveis externos ao ERP, é necessário licenciar o módulo "Plataforma de Integração" para o número máximo de postos em uso pelo cliente final. Este módulo já está incluído de base na linha Extended.

NameSpace

As classes da API encontram-se quase todas no namespace Eticadata.

Daqui em diante, quando não houver referência em contrário, assume-se que todos os tipos se encontram neste namespace.

B) O "ARRANQUE" DA APLICAÇÃO

Para começar a utilizar a API, é necessário inicializar um objeto com o tipo *EtiAplicacao*.

É a partir deste objeto que se irão depois obter referências para todas as operações sobre os dados do ERP.

```
Public Function InitializeEtiApp(ByVal applicationName As String, ByVal serverName As String, ByVal SystemDatabase As String, ByVal SqlLogin As String, ByVal SqlPwd As String, ByVal rptBasePath As String, ByVal sysToolsPath As String, ByVal licenseStatus As String, ByVal pAppVersion As String, ByVal serverUrl As String) As Boolean
```

PARÂMETROS:

strApp	Indica o nome da aplicação que está a inicializar o motor. Por regra, deverá ser Eticadata.Platinum.EtiConstantes.cAplicBackOffice
strServidor	Nome do servidor e instância Sql ao qual se pretende ligar.
strDataBaseSistema	Nome da base de dados de sistema à qual se pretende ligar
strLoginSrv	Nome do utilizador Sql.
strPwdSrv	*Password* do utilizador Sql
rptBasePath	Diretoria onde estão os *reports* do ERP. Poderá ser passada uma *String* vazia caso não seja conhecido.

sysToolsPath	Diretoria onde estão os ficheiros para importação de linhas de periféricos. Poderá ser passada uma *String* vazia caso não seja conhecido.
licenseStatus	Campo para uso interno. Deverá ser passada uma *String* vazia.
pAppVersion	Campo para uso interno. Deverá ser passada uma *String* vazia.
serverUrl	Url onde estão os componentes Web. Deverá ser passada uma *String* vazia, caso não seja conhecido.
Resultado	Indica se foi ou não possível inicializar a aplicação.

Além disso, é necessário efetuar o *login*. Isso é feito usando o método *Login* do objeto que acabamos de criar.

```
Public Function Login(ByVal userName As String, ByVal passWord As String) As LoginResult
```

Parâmetros:

strLogin	Nome do utilizador eticadata.
strPassword	*Password* do utilizador eticadata
Resultado	Indica se foi ou não possível fazer *login*.

Antes de fazer alguma operação, é ainda preciso abrir uma empresa e um exercício (embora neste último caso seja possível manipular entidades que estejam noutros exercícios).

```
Public Function OpenEmpresa(ByVal strCodEmpresa As String) As Boolean
```

Parâmetros:

strCodEmpresa	Indica o código da empresa que se pretende abrir (só o código, não o nome da base de dados).
Resultado	Indica se foi ou não possível abrir a empresa.

Public Function OpenExercicio(ByVal strCodigo As String) As Boolean

Parâmetros:

strCodigo	Indica o código do exercício que se pretende abrir.
Resultado	Indica se foi ou não possível abrir o exercício.

É também aconselhável abrir uma secção, embora isso não seja obrigatório para várias operações disponíveis. Além disso, é normalmente possível manipular os objetos de outras secções, mesmo abrindo uma secção.

Function OpenSeccao(ByVal strCodSeccao As String) As Boolean

Parâmetros:

strCodSeccao	Indica o código da secção que se pretende abrir.
Resultado	Indica se foi ou não possível abrir a secção.

A partir daqui, estamos prontos para utilizar a API para manipular dados do ERP.

C) SINGULARES E PLURAIS

Os objetos que permitem a manipulação de entidades do ERP estão normalmente divididos em 2 "categorias":

Os objetos plurais que representam o conjunto dos registos de determinada entidade e permitem a inserção, remoção, procura e carregamento de objetos singulares. Contêm também funções diversas, conforme as entidades. Estas classes normalmente têm o nome dos objetos singulares, mas acabando em "S" (por exemplo Clientes, Artigos, MovVendas, etc.). A referência para estes objetos é obtida a partir de um objeto do tipo *EtiAplicacao* devidamente inicializado.

Os objetos singulares que representam um registo de uma entidade e permitem a manipulação das propriedades desse registo, assim como operações que envolvam esse registo. As referências para objetos singulares, quer novos quer existentes, são criadas com recurso a um objeto plural da entidade pretendida.

D) MÉTODOS E PROPRIEDADES COMUNS

Entre as várias entidades há várias propriedades e métodos que são comuns, embora por vezes tenham ligeiras *nuances*.

- Propriedade Singular.IsNew

Esta propriedade indica se o objeto é novo e ainda não foi inserido na base de dados ou se representa uma entidade que foi carregada a partir da base de dados.

- Método Plural.GetNew

Permite obter uma instância de um objeto singular que ainda não está registado na base de dados. Enquanto não for feito um *update*, o objeto ainda não foi registado na base de dados. No caso dos movimentos que tem numeração, é calculado neste momento o número previsto do documento. No entanto, durante a gravação, é gerado o número definitivo do documento, pois podem ter ocorrido gravações de documentos do mesmo tipo no intervalo de tempo entre o GetNew e o Update de um objeto.

- Método Plural.Find

Permite procurar e carregar um objeto existente na base de dados. Como parâmetro tem de ser indicada a chave da entidade (cujo número de campos e respetivo tipo varia conforme a entidade). Caso não seja encontrado nenhum registo com a chave indicada, será devolvido um novo registo. Assim, o sucesso da operação de Find tem de ser verificado, acedendo à propriedade IsNew do objeto devolvido.

- Método Plural.Validate ou Singular.Validate

Antes de tentar fazer o Update dum objeto singular, é conveniente verificar se este se encontra íntegro, de forma a evitar pedidos inúteis à base de dados. Para isso, existe um método de validade (em algumas entidades existe no singular, noutras no plural), que permite saber se o objeto é válido ou não. Estes métodos devolvem também, conforme as entidades, uma lista de questões que podem ou não ser validadas com o utilizador. Pois, apesar de haver dados que não impedem a gravação do registo, denotam alguma incoerência dos mesmos (por exemplo: O número de contribuinte não é válido. Deseja continuar?). O método indica se o registo é ou não válido. Para ver o motivo do erro, pode consultar as propriedades EtiErrorCode e EtiErrorDescription e EtiErrorLine do objeto singular.

- Método Plural.Update

É na sequência deste método que os objetos são guardados na base de dados. Caso haja erros, são preenchidas as propriedades EtiErrorCode e EtiErrorDescription e EtiErrorLine do objeto singular.

Em alguns casos, o método lança uma exceção que deverá ser tratada pelos métodos normais.

- Método Plural.Delete

Este método permite remover uma entidade da base de dados. Normalmente, leva como parâmetro a chave do registo a remover. Antes de ser chamado, pode-se chamar o método PreDelete para verificar se é possível remover sem causar erros. Após a invocação do método,

podem-se consultar as propriedades EtiErrorCode e EtiErrorDescription e EtiErrorLine para saber se houve erros.

- Métodos Plural.MoveFirst, Plural.MoveLast, Plural.MovePrevious e Plural.MoveNext

Estes métodos permitem obter um determinado objeto singular relativamente a um outro. O primeiro ou o último objeto da lista, o anterior ou seguinte a um dado registo.

- Propriedade Singular.EtiErrorCode, Singular.EtiErrorDescription e Singular.EtiErrorLine

Permitem consultar o estado de erro de um objeto. Após uma operação de Validate, Update ou Delete, o EtiErrorCode está vazio em caso de sucesso, ou preenchido com um código em caso de insucesso. No case de insucesso, o EtiErrorDescription contém uma descrição do erro e o EtiErrorLine tem o número da linha onde ocorreu o erro, se este é relativo a uma linha da entidade. Nem todos os objetos têm esta última propriedade.

- Método Exists

Em alguns casos, para saber se determinado registo existe, está disponível o método Exists que apenas verifica se existe um registo com o código indicado, em vez de tentar carregar toda a estrutura de um objeto (que pode ser bastante complexa, em alguns casos) apenas para se consultar a propriedade IsNew.

E) CONSULTAS DE INFORMAÇÃO E/OU OBTENÇÃO DE LISTAS DE REGISTOS

A API disponibilizada pela eticadata não está muito vocacionada para a consulta de um determinado campo de um registo ou para a obtenção de listas (que podem ser extensas) de registos.

Assim, se num determinado algoritmo é necessário consultar apenas um campo de um registo, é preferível efetuar uma *query* diretamente à base de dados, retornando apenas o campo necessário.

Da mesma forma, se for preciso obter uma lista de registos, tipicamente para apresentar numa janela, ou para iterar, efetuando uma operação em cada entidade, é preferível efetuar uma *query* à base de dados, retornando os campos que se pretendem nas condições desejadas.

Para facilitar estas consultas, é possível obter as *connection strings* de ligação ao SqlServer, quer do sistema quer da empresa.

Estas estão disponíveis através das propriedades:

EtiApp.ActiveEmpresa.ConnectionStringNET

EtiApp.SysConnectionStringNET

(EtiApp representa uma variável do tipo EtiAplicacao)

F) PRINCIPAIS TABELAS

Para aceder aos plurais das principais tabelas, um objeto do tipo EtiAplicacao disponibiliza vários métodos:

- Tabelas – Permite aceder aos plurais das tabelas dos módulos Comercial e Contabilidade. As tabelas que são comuns a vários módulos encontram-se aqui (clientes, fornecedores, artigos, secções, tipos de documento, etc.).
- TabelasAutoGest – A mesma coisa para os módulos de Stands, Oficinas e Rent-a-car.
- TabelasGrh – A mesma coisa para o módulo de GRH.
- TabelasImo – A mesma coisa para o módulo de Investimentos.
- TabelasPOSRest – A mesma coisa para tabelas do POS e Gourmet.

Dado o elevado número de tabelas, não serão aqui descritas as que estão disponíveis. Aconselha-se o uso de uma ferramenta que tenha um *object browser* ou *intelisense* para observar todas as propriedades que estas classes disponibilizam.

G) PRINCIPAIS MOVIMENTOS

Para os movimentos, um objeto do tipo EtiAplicacao disponibiliza vários métodos:

- Movimentos – Permite aceder aos plurais dos movimentos dos módulos Comercial e Contabilidade.
- MovimentosAutoGest – A mesma coisa para movimentos de Stands, Oficinas e Rent-a-car.
- ProcessamentosGrh – A mesma coisa para processamentos do módulo de Recursos Humanos.
- ProcessamentosImo – A mesma coisa para os processamentos do módulo de Investimentos.

H) MANIPULAÇÃO DE ENTIDADES

As classes singulares representam uma entidade do sistema, seja um registo simples, como um meio de expedição, por exemplo, ou um mais complexo, como um movimento de venda. Naturalmente, dependendo do tipo de entidade, existirão mais ou menos propriedades

intrínsecas ao tipo de entidade que estamos a manipular. Enquanto no primeiro exemplo o objeto singular pouco mais terá de interessante que o código ou a descrição, no segundo caso, teremos uma estrutura de cabeçalho, outra de linhas, que por sua vez se subdividem em linhas de satisfação de encomendas, linhas de números de série, linhas de componentes, que por sua vez se subdividem em mais linhas, e cada uma destas subdivisões tem as suas próprias propriedades.

Para alterar um objeto, devemos alterar as suas propriedades. Mas, por vezes, isto não é suficiente. Em seguida, abordaremos algumas especificidades de campos e métodos especiais para efetuar algumas operações.

- Altera

Normalmente quando queremos alterar a informação de uma dada entidade, atribuímos o valor pretendido à propriedade respetiva. No entanto, há casos em que queremos que o sistema faça um pouco mais. Por exemplo, quando alteramos a entidade duma venda, poderemos querer que a venda assuma as informações do cliente (meio de pagamento, moeda, armazém, etc.). Para isso, não basta atribuir a propriedade. Existe um método no objeto principal do Movimento de Venda que deverá ser chamado após a atribuição da entidade e que indica que o cliente foi alterado e que deverão ser calculadas as suas informações. Estes métodos também verificam se há eventos de utilizador a executar e tratam de os executar (neste caso em concreto, seria executado o evento de alteração de entidade).

Esses métodos chamam-se Altera<Campo>, e o número e tipo de parâmetros varia conforme o campo. Por vezes, devolvem informações que podem ser úteis quanto aos procedimentos a executar em seguida. Por exemplo, ao inserir um artigo numa linha de um documento de venda, poderá ser devolvida a indicação se há stock disponível para o artigo em questão.

- Entidades com Linhas / Listas

Em grande parte dos objetos existe informação que não tem um número predefinido. Por exemplo, os movimentos de venda têm uma lista de linhas. Esta lista pode ser acedida através da propriedade Linha, que é indexada ao número da linha que se pretende consultar.

Para saber quantas linhas existem, existe o método CountLin. Para cada lista existe um correspondente CountLin – um para artigos, outro para pagamentos, etc.

As linhas começam sempre em 1.

Para adicionar uma linha, temos o método AddLin. Como parâmetro, deverá ser passado o número da linha que pretendemos inserir. Tal como o CountLin, também há um AddLin para cada lista existente.

- Campos Indexados de Linhas

Em alguns casos, principalmente nas tabelas, as listas são implementadas de uma forma ligeiramente diferente. Existem de igual forma as Propriedades CountLin e AddLin, no entanto, em vez de se aceder a um objeto do tipo Linha, que por sua vez tem as propriedades de uma linha, estas propriedades estão no objeto principal, sendo indexadas à linha correspondente. Por exemplo, os artigos têm linhas de preços. A classe Artigo disponibiliza os métodos AddLinPrecos e CountPrecosVenda. No entanto, não há um objeto LinhaPrecos. A classe Artigo disponibiliza as propriedades Numero, Preco, PrecoAnterior1, etc., que são indexadas à linha que pretendemos consultar/alterar.

02 | PROGRAMAÇÃO DE REGRAS E EVENTOS

Na programação de regras e eventos, a API discutida no capítulo anterior está toda disponível. Naturalmente que, uma vez no ERP, já foi feita a inicialização dos objetos necessários, pois o utilizador já se autenticou, abriu a empresa, etc.

Em alguns casos, são passados para as regras e eventos mais alguns objetos que permitem saber o contexto da operação (transação, ligação à base de dados, etc.).

Vamos neste capítulo tentar abordar algumas formas de tirar o máximo partido da programação que é possível fazer dentro das regras e eventos para personalizar o ERP, levando-o a cumprir objetivos para os quais não tem suporte de raiz.

A) OPERAÇÕES COMUNS

Um evento não é mais que um método que é chamado quando algo predeterminado acontece.
Estes métodos têm de ser programados na janela disponível para o efeito no administrador.

Tipicamente, num evento podemos consultar as propriedades de um determinado objeto, socorrendo-nos do motor de integração (que é passado como parâmetro nas funções de eventos) para outras operações sobre outros registos, que de alguma forma estejam relacionados com o que pretendemos fazer.

- Operações sobre Campos Adicionais

  Para operar sobre os campos adicionais das entidades que os suportam, está disponível uma propriedade CampoAdicional, que é indexada pelo nome do campo adicional. Assim, para ler o valor atual de um campo adicional num objeto de clientes, poderíamos usar:

```
Dim bln = CBool(Cliente.CampoAdicional("CA_Permite"))
```

  De igual forma, para escrever deveremos usar:

```
Cliente.CampoAdicional("CA_Permite") = False
```

No caso dos movimentos, os campos adicionais podem ser do cabeçalho ou das linhas.

Os campos do cabeçalho são acedidos de igual forma:

MovVnd.Cabecalho().CampoAdicional("CA_NumOrdem") = 10

No caso das linhas, que têm um número de campos adicionais predefinido, tem de se aceder pelo índice numérico do campo adicional:

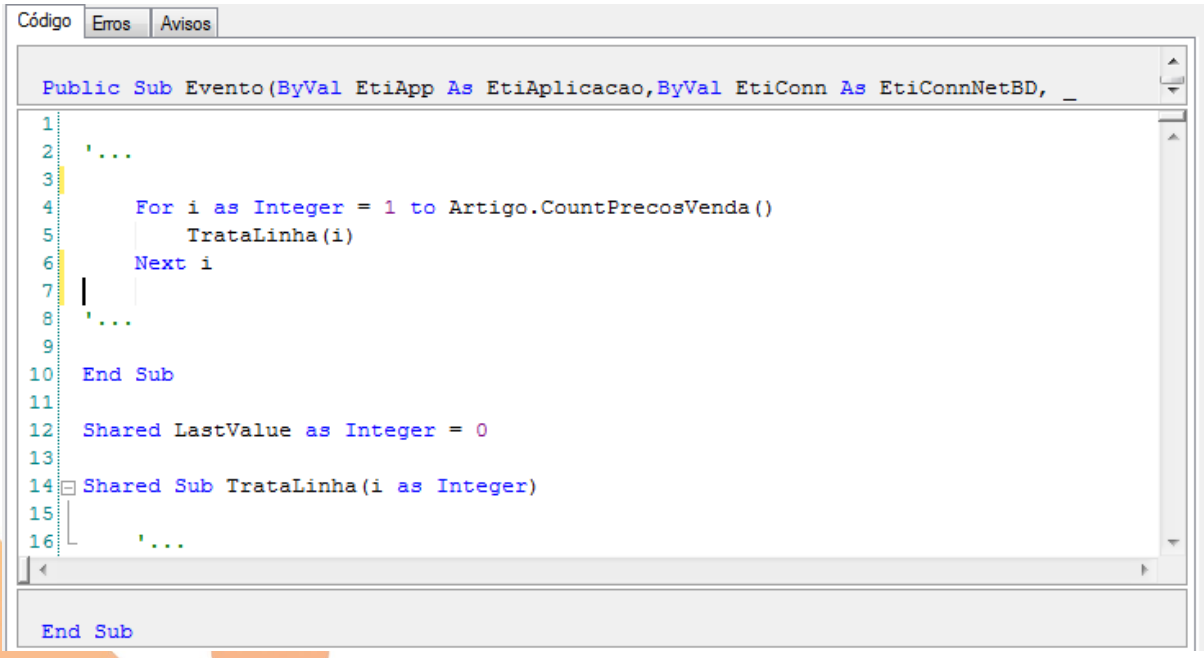
MovVnd.Linha(1).CampoAdicional(1) = DateTime.Now

B) CRIAÇÃO DE FUNÇÕES/PROCEDIMENTOS

Por vezes, no meio de um evento, poderá ser útil criar procedimentos ou funções que ajudem a estruturar o código desenvolvido.

Mas como fazer isto, se o editor de código dos eventos só disponibiliza uma janela e esta tem já o princípio e o fim de uma função?

Se soubermos que todo o código que está na janela vai ser compilado numa classe estática, podemos terminar o método principal e iniciar outros, desde que o código seja válido.



Da mesma forma, é possível criar classes internas, variáveis que guardam o seu valor entre invocações, etc.

C) ALGUMA INTERFACE COM O UTILIZADOR

Da mesma forma que se podem criar funções e procedimentos adicionais, pode-se criar alguma interface com o utilizador, criando classes que herdam de System.Windows.Forms.Form e são manipuladas em código.

Código | Erros | Avisos

```
Public Sub Evento(ByVal EtiApp As EtiAplicacao,ByVal EtiConn As EtiConnNetBD, _
15  End Sub
16  '...
17  Private Class Janela
18      Inherits Form
19
20      Private WithEvents btn as New Button
21      Friend WithEvents txt as New TextBox
22
23      Sub InitializeComponents()
24          txt.Left = 0
25          txt.Top = 0
26          txt.Width = 200
27          txt.Visible=true
28          btn.Text = "Ok"
29          btn.Left = 220
30          btn.Top = 0
31          btn.Width = 50
32          btn.Visible=true
33          me.Width= 330
34          me.Height = 80
35          me.Controls.Add(txt)
36          me.Controls.Add(btn)
37      End Sub
38
39      Sub btn_clik(sender as Object, args as EventArgs) Handles btn.Click
40          Me.Close()
41      end Sub
42
43  End Class
44
```

Estas janelas podem ser usadas para pedir mais dados ao utilizador, para apresentar o progresso de uma operação mais longa, etc.

Atenção – Os eventos e regras de validação são executados mesmo quando o motor de integração está a ser executado por outras aplicações que não o ERP. Por exemplo, se for programado num evento Gravar Artigo a apresentação de uma janela, se houver um qualquer processo, por exemplo ligado a *site*, que tente gravar artigos, provavelmente vai apresentar um erro, pois não irá conseguir apresentar a janela.

D) REFERÊNCIA A DLLS EXTERNAS

Apesar de ser possível programar janelas totalmente em código, não é prático programar objetos "visuais" (que vão ter uma representação visual para o utilizador) em código, sem as facilidades oferecidas por um moderno IDE. Mesmo quando as operações a implementar implicam uma extensão e organização de código que obriga a uma organização em várias classes, acesso a outros sistemas, etc., é possível a referenciação de Dlls feitas num qualquer IDE.

Estas Dlls deverão ter as referências para as Dlls de base do motor de integração (ver capítulo 1) para poderem manipular os objetos da eticadata (se for necessário).

Para adicionar uma Dll externa, deve aceder ao menu Admin, Customização e Assemblies. Nesta janela, poderão ser inseridas ou removidas Dlls que serão usadas para a compilação dos eventos, caso estejam marcadas para tal.

Estas Dlls vão ser disponibilizadas aos clientes *desktop* ligados a um servidor Web na próxima inicialização dos mesmos.

Ao serem carregadas, são automaticamente feitos os *imports* de todos os namespaces que disponibilizam, de modo a que se possam usar os tipos que expõem.

A utilização é feita como se de um qualquer outro tipo se tratasse.

```
dim obj as new OMeuTipoExterno()

obj.ActualizaOutroSistema(Artigo.Codigo, Artigo.Descricao)

obj.ApresentaJanelaEActualizaArtigo(Artigo)
```

03 | INTEGRAÇÃO DE NOVAS FUNCIONALIDADES

É muitas vezes necessário complementar as janelas do ERP com operações específicas para um determinado cliente. Acrescentar menus que o utilizador chama, que abrem novas janela e que manipulam a entidade do ERP e/ou de outros sistemas. Neste capítulo, vamos tentar abordar as formas de o fazer.

A) INTRODUÇÃO AO MODELO CAB

O ERP eticadata é uma aplicação modular, isto é, o executável que o utilizador chama não sabe, à partida, quais as janelas que vão ser usadas. Isto depende dos módulos licenciados e das permissões do utilizador. Assim, as Dlls que são carregadas são as necessárias pelos menus.

Além disso, as Dlls não se conhecem entre si, nem sabem os tipos e funcionalidades que as outras expõem.

Para atingir estes objetivos, a eticadata utiliza uma versão customizada do *Component Application Block*, disponibilizado pela Microsoft.

A comunicação entre os vários componentes é feita com recurso a 4 mecanismos:

- Estado

  É possível, em qualquer momento, guardar, num dicionário indexado por uma chave, um qualquer objeto. Quando em determinado contexto queremos passar informação a outra entidade, podemos guardar essa informação com uma chave que ambas conhecem e invocar essa entidade. A entidade "destino" acederá à mesma chave e obterá as informações que lhe enviamos. Um exemplo típico de informação que é colocada no estado é o código da entidade que queremos editar, quando se quer abrir uma janela de edição de entidades.

- Serviços

  Os serviços são parecidos com o estado. No entanto, não são indexados por uma chave, mas sim por um tipo. A qualquer momento, quando é colocado um objeto na coleção de serviços, este pode ser acedido posteriormente, simplesmente dizendo que se quer o serviço de um determinado tipo. Normalmente, está aqui o motor de integração que é usado pelo ERP, entre outros objetos necessários em toda a aplicação.

- Comandos

  Os comandos são a forma mais simples de invocar uma ação. Ao serem associados a um ou mais métodos, sempre que se manda executar um comando, o motor CAB executa todos os métodos que estão associados ao comando.

  Por exemplo, ao carregar os menus, é associado um método ao comando definido por um menu. Sempre que o utilizador clica no menu, é executado o comando e, internamente, é encontrado o método que deverá ser executado. Os menus, ao serem carregados, não sabem quem vai responder aos comandos. Apenas sabem que os têm de invocar.

- Eventos

  Os eventos são semelhantes aos comandos, mas podem passar e receber um contexto. Tal como os comandos, são registados os métodos que vão responder a eventos e, quando se quer invocar um evento, o motor CAB procura todos os métodos que subscreveram o evento e executam-no.

- Atributos

  A associação de Comandos e Eventos pode ser feita através de linhas de código, mas é mais cómodo efetuá-la através de atributos. Assim, quando pretendemos que um método responda a um comando, decoramos esse método com um atributo que diz ao motor CAB que esse método deverá ser executado quando alguém invocar esse comando. De igual forma, podemos decorar propriedades para serem automaticamente injetadas a partir de Serviços ou Estado.

- Carregador de Módulo

  Naturalmente, os atributos não são todos carregados quando se carrega um *assembly* com vários tipos.

  O motor CAB vê quais os tipos que herdam do tipo ModuleInit. Nesse momento, cria uma instância desse tipo e analisa todos os seus métodos e propriedades, procurando os atributos conhecidos e efetuando as ações necessárias.

  A partir daqui, todas as classes que forem adicionadas ao motor têm o mesmo tratamento.

B) REFERÊNCIAS NECESSÁRIAS

Para criar uma Dll que seja reconhecida pelo motor CAB, é necessário que esta tenha referências às Dlls distribuídas com a aplicação:

Eticadata.Infrastructure.dll

Será ainda aconselhável importar os namespaces seguintes às classes que se quer que interajam com o motor CAB:

```
Imports Microsoft.Practices.CompositeUI.Services

Imports Microsoft.Practices.ObjectBuilder
```

C) PREPARAR UMA DLL PARA SER CARREGADA

Para carregar uma Dll, é necessário criar uma entrada na *ribbon* e associar esta a um comando. Além disso, é necessário indicar ao sistema que há uma Dll a ser carregada, no menu Admin, Customização, Assemblies.

Desta forma, o ERP vai carregar o *assembly* no arranque, irá procurar todas as classes que herdam do tipo ModuleInit, e irá instanciá-las. Irá também analisar os métodos que estão decorados com o atributo CommandHandler e associá-los à execução do comando, fazendo o mesmo com os eventos com o atributo EventBroker.EventPublication.

Além disso, é possível fazer o Override ao método Load para executar código aquando do carregamento do módulo.

Uma variável importante para se obter é o WorkItem. Este representa uma entidade que foi carregada pelo motor CAB. A partir desta variável, pode-se aceder a todo o sistema CAB carregado.

D) INJEÇÃO DE SERVIÇOS E ESTADO

Para obter uma referência aos serviços disponíveis no motor CAB, pode-se declarar uma propriedade e decorá-la com o atributo ServiceDependency. Ao criar o objeto, o motor CAB irá procurar na coleção de Serviços um serviço com o tipo da propriedade e irá atribuir a propriedade com a instância que está guardada.

Por exemplo, para obter uma referência ao motor Eticadata, podemos fazer:

```
Private myEtiApp As EtiAplicacao


<ServiceDependency()> _
Public Property EtiApp() As EtiAplicacao
 Get
   Return myEtiApp
 End Get
 Set(ByVal value As EtiAplicacao)
   myEtiApp = value
 End Set
   End Property
```

Alternativamente, podemos obter esta referência se tivermos uma referência a um WorkItem. Para obter uma referência ao motor eticadata podemos fazer:

```
Dim myEtiApp As EtiAplicacao = myWorkItem.Services.Get(Of EtiAplicacao)()
```

NOTA: O motor de integração eticadata não deverá ser guardado em variáveis da classe ModuleInit, pois sempre que a empresa for aberta, esta variável será "reciclada" e a instância guardada deixará de ser válida.

De igual forma, é possível obter a informação que está na coleção de estados, desde que seja indicada a respetiva chave de indexação:

```
    Private myCod As String

    <State("Código")> _

    Public Property Cod() As String

Get

   Return myCod

End Get

Set(ByVal value As String)

   myCod = value

End Set

    End Property

    Dim Cod As String = myWorkItem.State("Código").ToString()
```

E) INVOCAR E RESPONDER A COMANDOS E EVENTOS

Para invocar um comando, que terá uma ação por parte do sistema, devemos fazer:

```
myWorkItem.Commands(CommandsArtigos.EditarArtigos).Execute()
```

Substituimos a constante CommandsArtigos.EditarArtigos por um comando que seja conhecido e que inicie a ação pretendida. Neste caso, será aberta a janela Editar Artigos.

No caso dos comandos eticadata, estão normalmente definidos em Constantes que começam por Eticadata.Common.Commands<item>, definidos na Dll Eticadata.Controls.Dll.

Para que um determinado método seja executado em resposta a um comando, é necessário que sejam cumpridos dois requisitos:

a) O método tem de ter a assinatura normal de um evento.
b) O método tem de estar decorado com o atributo CommandHandler.

Por exemplo:

```
<CommandHandler("A_Minha_Accao")> _
Public Sub OnMinhaAccao(ByVal sender As Object, ByVal e As EventArgs)
'...
End Sub
```

Neste caso, sempre que for invocado o comando "A_Minha_Acao", o motor CAB irá executar o método OnMinhaAcao.

No caso dos eventos, deve-se proceder de forma semelhante:

Publicar um Evento:

```
<EventBroker.EventPublication("MyEvent", EventBroker.PublicationScope.Global)> _
Event myEvent(ByVal o As Object, ByVal e As MyEventArgs)
```

Onde for necessário, fazer o *raise* do evento.

Responder a um evento:

```
    <EventBroker.EventSubscription("MyEvent", EventBroker.ThreadOption.UserInterface)> _
    Public Sub OnMyEvent(ByVal sender As Object, ByVal e As MyEventArgs)
'...
    End Sub
```

De notar que os parâmetros entre a publicação e a subscrição têm de ser iguais.

C) ABERTURA DE JANELAS

Uma tarefa normal num módulo CAB é a abertura de janelas. Estas podem ser abertas de diversas formas.

Usando o motor CAB, podem-se usar todas as facilidades de atributos e propriedades atrás descritas. Neste caso, as janelas não são programadas como *Forms* mas como *UserControls*. Cabe ao motor CAB a criação da janela onde o *UserControl* irá ser colocado em *runtime*.

Para simplificar o trabalho, a eticadata criou uma classe genérica para permitir de forma fácil abrir *UserControls* baseados no CAB.

Em última análise, é possível obter uma referência à janela principal e mostrar uma janela "normal", que foi devidamente inicializada pelos meios tradicionais do .NET.

- Usando o Motor CAB

Para abrir uma janela, é necessário criar uma classe que herda de workItem e, nesta, criar um método para apresentar a janela recorrendo aos métodos do motor CAB.

Esta classe deverá ter, mais ou menos, a seguinte estrutura:

```
Public Class WorkItemClass

        Inherits WorkItem

  Public Sub MostraJanela()

    Dim theWindow = Me.Items.AddNew(Of myJanelaUserControl)("IdJanela")

        Dim mainWorkspace As IWorkspace =
        RootWorkItem.Workspaces(WorkspacesConstants.MAIN_WORKSPACE)

        Dim spi = New WindowSmartPartInfo With {.Title = "A minha Janela", .Location = New
        Point(10, 10)}

     mainWorkspace.Show(theWindow, spi)

       End Sub

End Class
```

Para a carregar, será necessário, como resposta a um comando, por exemplo, fazer:

```
myWorkItem.WorkItems.AddNew(Of WorkItemClass)().MostraJanela()
```

- Usando a classe genérica GWorkItem

Para evitar ter de criar uma classe Workitem apenas para conter um método para abrir janelas, a eticadata desenvolveu um método genérico que trata destas aberturas. Esta classe trata ainda do posicionamento e dimensão da janela, colocando-a na última posição e tamanho em que se encontrava quando foi fechada pelo utilizador.

Para abrir uma janela (UserControl) usando esta classe, deve-se fazer, como resposta a um comando, por exemplo:

```
GWorkItem.ShowWindow(of
myJanelaUserControl)(myWorkItem,"MyCommandName",blnModal,blnFixed)
```

Sendo blnModal a indicação de se a janela deverá ser modal ou não e blnFixed a indicação de se será redimensionável ou não.

- Janelas Normais

Em última análise, é possível abrir janelas normais, criando uma variável de um tipo que herda de Windows.Forms.Form, atribuir-lhe o MDIParent e mostrá-la:

```
Dim myForm As FormHeir

Dim mainWorkspace As IWorkspace =
myWorkItem.Workspaces(WorkspacesConstants.MAIN_WORKSPACE)

myForm.MdiParent = CType(mainWorkspace, WinForms.MdiWorkspace).ParentMdiForm

myForm.Show()
```

Naturalmente que, se a janela não tiver de ser MdiChild, então é só mostrá-la.

D) INVOCAR OUTRAS JANELAS

Para invocar as janelas do ERP, é apenas necessário executar os comandos respetivos.

Algumas janelas do ERP suportam o posicionamento, isto é, é possível indicar o registo no qual queremos que uma janela se posicione.

Para isso, é necessário indicar no Estado GlobalState.Posicao **um *array* de *strings*** com os valores chave do registo. Este pode variar entre uma única posição com um campo simples (por exemplo os artigos ou clientes), mais que um campo (os funcionários, ou contas da contabilidade, têm o código do exercício e o código do funcionário ou conta), os movimentos comerciais que levam 4 campos (tipo de documento, número, exercício e secção) e os movimentos de Autogest com (secção, tipo de documento, exercício e número).

E) IMPRIMIR/EXPORTAR DOCUMENTOS

Por vezes, é necessário imprimir documentos comerciais. Para isso, está disponível na Dll Eticadata.Views.Reports uma classe que nos permite fazê-lo.

Depois de devidamente inicializada a classe, poderá ser chamado o método que Emite o documento. Este método permite a pré-visualização, a impressão e o envio de emails, caso o tipo de documento esteja configurado para isso aquando da impressão.

Para pré-visualizar ou imprimir documentos, deveremos primeiro inicializar uma variável do tipo Eticadata.Views.Reports.ReportsGcePOS.

Esta variável deverá ser inicializada com o construtor:

```
Public Sub New(ByVal oEtiApp As EtiAplicacao, ByVal workItem As WorkItem, ByVal uiUtils As UIUtils, ByVal diretoriaRpt As String, ByVal blnSilentMode As Boolean)
```

Em seguida, deveremos usar o método EmiteDocumentoCust para pré-visualizar ou imprimir os documentos.

```
Public Sub EmiteDocumentoCust(ByVal intFrontOffBackOff As Short, ByVal lngPerfilPerifericos As Integer, ByVal pTpDocEmitir As TpDocumentoAEmitir, ByVal blnImprimir As Boolean, ByVal strCodExercicio As String, ByVal strAbrevTpDoc As String, ByVal lngNumero As Integer, ByVal intConfigImpressao As Short, ByVa
```

l intGravacao As Short, ByVal intMovimento As Short, ByVal pEtiapp As EtiAplicacao, ByVal pgDirectoriaRpt As String, ByVal pNomeFicheiro As String)

Por exemplo:

```
Dim myReportsGcePOS As New Eticadata.Views.Reports.ReportsGcePOS(myEtiApp, myWorkItem, myUiUtils, myEtiApp.DirectoriaRpt, False)

myReportsGcePOS.EmiteDocumentoCust(2, -1, TpDocumentoAEmitir.Encomendas, False, "2012", "ENCFO", 1, 0, 1, myEtiApp, myEtiApp.Ambiente.DirectoriaRpt, String.Empty)
```

Caso não sejam conhecidos os valores de myWorkItem e myUiUtils, estes podem ser passados a *null* (Nothing em VB.NET).

Se quisermos obter um ficheiro com a representação do *report* em PDF, deveremos proceder de uma forma parecida. Desta vez, temos de inicializar uma variável do mesmo tipo, mas com o construtor:

```
Public Sub New(ByVal oEtiApp As EtiAplicacao, ByVal diretoriaRpt As String, ByVal export As ExportWebFormat)
```

Em seguida, devemos chamar o método:

```
Public Function EmiteDocumento(ByVal pTpDocEmitir As TpDocumentoAEmitir, ByVal blnImprimir As Boolean, ByVal strCodExercicio As String, ByVal strAbrevTpDoc As String, ByVal lngNumero As Integer, ByRef arrayFormulas As Object,  ByRef intGravacao As Short, ByRef intMovimento As Short, ByRef strCodSeccao As String, ByVal pNomeFicheiro As String, ByRef FileNotFound As Boolean) As Byte()
```

Este método devolve um *array* de bytes com o PDF correspondente ao documento que pretendemos emitir.

Por exemplo:

```
Dim myReportsGcePOS As New Eticadata.Views.Reports.ReportsGcePOS(myEtiApp,
myEtiApp.DirectoriaRpt, eticadata.Views.Reports.ReportsGcePOS.ExportWebFormat.PDF)
    Dim bt = myReportsGcePOS.EmiteDocumento(TpDocumentoAEmitir.Encomendas, False, "2012",
"ENCFO", 1, Nothing, 0, 1, "SEC1", String.Empty, False)
    Dim fileName = System.IO.Path.GetTempFileName() & ".pdf"
    System.IO.File.WriteAllBytes(fileName, bt)
    Process.Start(fileName)
```

F) REGISTO DE DOCUMENTOS DE VENDAS VS. CERTIFICAÇÃO

Há cenários em que pretendemos que, ao registar documentos de vendas no ERP estes sejam assinados pela infraestrutura da eticadata. Neste caso, a responsabilidade de geração da assinatura é da eticadata, e o registo e impressão do documento a apresentar ao cliente são feitos pelo ERP da eticadata.

Para poder registar um documento que seja assinado, devem estar reunidas as seguintes condições:

- Empresa sujeita a certificação:
  - País sede em Portugal.
  - Não é de formação.
- Ter o módulo "Plataforma de Integração com Certificação".
- Tipo do Documento ser para apresentar ao cliente.
- Secção não ser para documentos Externos nem Manuais.
- Tem que ser um documento novo: *IsNew = True*.
- Chamado o código da eticadata: myEtiApp.Movimentos.MovVendas.Update(...).
- Não esteja assinado.
- Não seja proveniente de Auto-Pré-Venda: *OrigemAutoPreVenda = False*
- Não seja proveniente de integração: *IntegracaoOffLine = False*

Cumpridas estas condições, o documento será assinado na 1ª gravação.

Para inserir um documento no ERP V18, que tenha sido assinado por uma aplicação externa, será necessário cumprir as mesmas condições, exceto a que diz respeito à secção, que passa a:

- Secção é para documentos Externos.

Neste caso, antes de gravar deve ser invocado o método AlteraInfoCertificacao com os parâmetros:

a) Num. Certificação – Número de certificação do software que gerou a assinatura.
b) Hash Control – Usado para a assinatura.
c) Hash – Hash calculado para o documento.