Treinamento OnPlace

Rev.3

Modelagem do banco de dados

- Assim como outros partes do aplicativo a modelagem foi feita visando ser genérica em vários aspectos.

- Padrão de nomenclatura:
  - As tabelas sempre começam com o prefixo ONP (abreviação de *OnPlace*).
  - **Prefixo seq:** Tipo *numeric* com quantidade de dígitos variável para cada situação. Exemplo: seq_fatura.
  - **Prefixo cod:** Tipo *varchar* contendo um código que pode misturar letras e número. Exemplo: cod_hidrometro.
  - **Prefixo ind:** Sempre é um tipo *varchar* com tamanho de 1 a 3 caracteres. É abreviação de indicativo que pode ser tipo booleano ou ser um conjunto predefinido de valores. No caso de ser tipo booleano os valores usados são "S" (Sim) e "N" (Não). Exemplo: ind_impresso (booleano) ou ind_situacao_movimento.
  - **Prefixo val:** Tipo *numeric* com casas decimais que no aplicativo é traduzido como um tipo double. Exemplo: val_valor_faturado.
  - **Prefixo des/nom:** Uma string com quantidade variável de caracteres (*varchar*). Exemplo: nom_cliente, des_endereco_alternativo.
  - **Prefixo dat:** Representa uma data e hora. Exemplo: dat_leitura_anterior, dat_vencimento.

- As tabelas com sufixo "DIADEMA" são as tabelas que guardam informações especificas da SANED.

- **Tabelas principais:**
  - **ONP_MATRICULA:** Guarda informações sobre o cliente.
  - **ONP_MOVIMENTO:** Guarda informações sobre a leitura anterior, consumo médio, medidor que a ligação usa e também o valor da leitura entrada em campo entre outras informações. Esta sempre ligado a uma matricula.
  - **ONP_MOVIMENTO_CATEGORIA:** Informações das categorias que estão ligadas ao movimento.
  - **ONP_MOVIMENTO_TAXA:** Informações das categorias que estão ligadas nas categorias de um movimento.
  - **ONP_MOVIMENTO_OCORRENCIA:** Tabela preenchida pelo agente de campo com as anormalidades constatadas na hora da leitura.
  - **ONP_SERVICO_FATURA:** Contém vários registros por movimento especificando serviços a serem faturados na hora de calcular a conta.
  - **ONP_MENSAGEM_MOVIMENTO:** Contém mensagens que aparecem na conta impressa.
  - **ONP_AVISO_DEBITO:** Cada matrícula contem um aviso de débito e este contém vários registros na tabela ONP_FATURAS_AVISO com valores que o cliente não esta devendo.
  - **ONP_HISTORICO:** Histórico de leituras para todas as matriculas do roteiro.
  - **ONP_FATURA:** Contém valor total da conta e mais várias informações copiadas de movimento. Esta tabela também tem registros referentes a segundas vias da matriculas do roteiro. Com exceção das segundas vias, as faturas criadas no coletor sempre vão ter um movimento associado a elas e o sequencial de identificação (seq_fatura) sempre será zero.

- **ONP_FATURA_CATEGORIA:** Somatório dos valores das taxas que foram calculadas. Uma categorias tem varias taxas.
- **ONP_FATURA_TAXA:** Guarda informações do consumo faturado, valor calculado e valor faturado. Uma taxa tem varias tarifas associadas a ela, mas somente as tarifas adequadas são usadas para fazer o calculo.
- **ONP_FATURA_SERVICO:** Contém vários registros por movimento especificando serviços a serem faturados na hora de calcular a conta.
- **ONP_PARAMETRO_RETENCAO:** Lista de parâmetro usados para verificar se a conta deve ficar retida para análise.
- **ONP_QUALIDADE_AGUA:** Informações que são impressas na conta sobre a qualidade da agua na zona na qual a matricula se encontra.
- **ONP_REFERENCIA_PENDENTE:** Guarda a lista de segundas vias disponíveis para impressão.
- **ONP_MATRICULA_ALTERACAO:** As alterações cadastrais feitas pelo agente de campo ficam armazenadas nesta tabela.

OnPlaceLoader

- Cria no computador uma base de dados (arquivo com extensão *sdf*) que é usada no *OnPlace*. Foi moldado de forma a permitir fácil alteração entre banco de dados (exemplo: *SqlServer* para *Oracle*) desde que exista uma implementação do *ADO.NET* para o banco de dados em questão.

- Controlador de carga é chamado de *ControladorDownload* e o controlador de descarga é chamado de *ControladorUpload*.

- Pela linha de comando é especificado os servidor, nome, usuário e senha do banco de dados entre outras informações.

- Usa o arquivo "metaDados.sql", que deve estar na mesma pasta do *OnPlaceLoader*, para criar a base de dados do *OnPlace*.

- Usa "BuildTask.dll" para incrementar o número da versão a cada compilação.

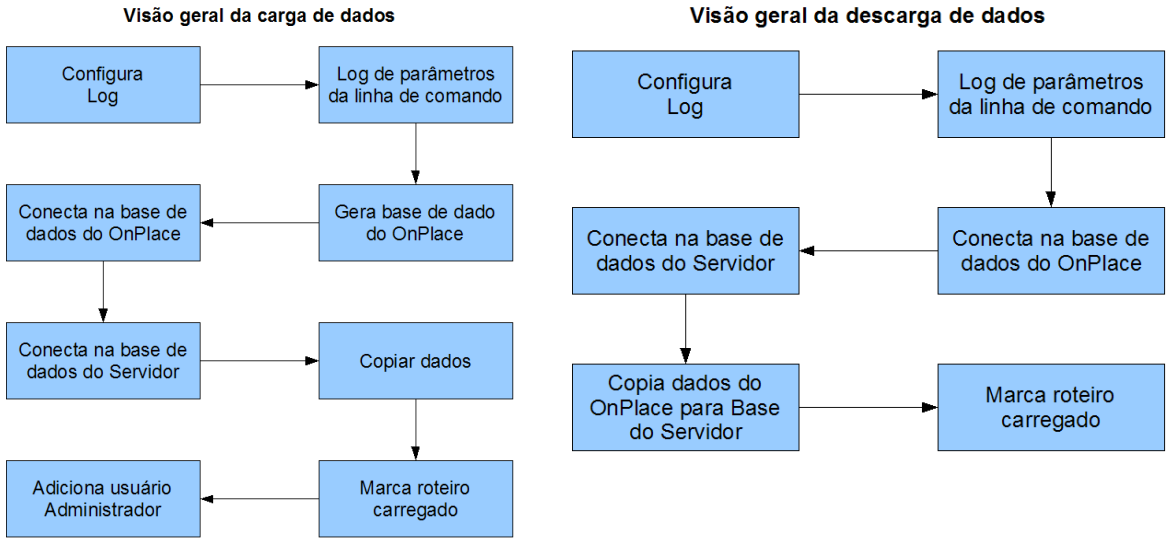
- Os controladores de carga e descarga são executados em uma *thread* diferente da *thread* do *FormLog*.

- Para funcionar é necessário que o *Sql Server Mobile* versão 3.1 esteja instalado no computador.

- Sempre gera um arquivo de *log* na mesma pasta do aplicativo com todas as ações que foram executadas. Erros durante o processo também aparecem neste arquivo.

- Caso tenha algum erro durante o processo de carga ou descarga, o programa fica aberto. Se tudo tiver dado certo o programa fecha sozinho ao final do processo.

- **Visão geral do processo de carga e descarga de dados:**



- **Classe *Singleton*:** Implementa o *design pattern singleton* na forma de um *generic*.

- **Classe *CommandLineParser*:** Herda *Singleton* e usando uma classe feita por terceiros faz o *parse* da linha de comando fornecida quando o aplicativo foi executado.

- **Classe *Controlador*:** Classe abstrata que tem funcionalidades comuns entre o controlador de *upload* e o controlador de *download*. Implementa a interface *IControlador*. Usa as interfaces do *ADO.NET*.

- **Classe *ControladorDownloader*:** Classe herda *Controlador* implementado alguns métodos abstratos. Extrai informações da base de dados do servidor e insere estas informações na base de dados do *OnPlace*.
  - **Método GerarBaseMovel:** usando o arquivo "metadados.sql" executa vários comandos "*create table*" a cada vez que acha um "*go*".
  - **Copia tabela inteira**: o método CarregarTabela fornece uma jeito de carregar todo o conteúdo de uma tabela para a base de dados do *OnPlace*.
  - **Copia usando *Query***: Executa uma *query* especifica no banco de dados do servidor e faz a cópia do resultado desta *query* para a base de dados do *OnPlace*.
  - A copia em si das informações é feita usando métodos de classes e interfaces do *ADO.NET*. Um *IDataReader*, contém o resultado de uma *query,* e um *DataTable*, vai receber os dados.

- **Classe *ControladorUploader*:** Classe herda *Controlador* implementado alguns métodos abstratos. Extrai informações da base de dados do *OnPlace* e insere estas informações na base de dados do servidor.
  - Somente algumas tabelas tem seu conteúdo alterado e assim somente essas tabelas são descarregadas.
  - As tabelas do servidor são carregas, atualizadas com o conteúdo vindo do *OnPlace* e então as alteração são "comitadas*"*.

- **Classe *FormLog*:** Fornece uma forma visual em "tempo real" de ver o que o programa esta fazendo. A janela pode ser escondida especificando um parâmetro na linha de comando.

StrategosAPI

- Acesso ao banco de dados por mapeamento de atributos de classes e colunas do banco de dados.
- Gera uma DLL que pode ser usada por outros projetos.
- Usa "BuildTask.dll" para incrementar o número da versão a cada compilação.
- **Custom Attributes em C#:** Ajuda a definir informações adicionais em classes, métodos, etc. Pode alterar ajudar a alterar o comportamento ou definir um comportamento.
- **Classe *TableAttribute*:** Ajuda a mapear uma classe para uma tabela do banco de dados, pode-se definir o nome da tabela na declaração do atributo ou deixar que a API use o padrão de nomenclatura dela. Para questões de performance é sempre aconselhável colocar o nome da tabela.
- **Classe *ColumnAttribute*:** Mapeia propriedades de uma classe para colunas da tabela que a classe representa no banco de dados. Assim como o *TableAttribute* pode-se especificar o nome ou deixar que a API use seu padrão de nomenclatura. Para questões de performance é sempre aconselhável colocar o nome da coluna.
- **Classe *Column*:** Funciona como um cache de informações para acesso ao propriedades de classes. Evita o uso de muito *reflection*.
- **Interfaces *IColumn*, *ICommand* e *IConnection*:** Utilizadas para deixar a camada de acesso ao banco independente do banco de dados que esta sendo usado.
- **Classe *CommandCE*:** Representa a implementação de um comando a ser executado no banco de dados do *SQL Server Compact Edition*. O comando ao qual a classe representa implementa as interfaces do *ADO.NET Data Provider*. Suporta parâmetros, execução de comandos escalar, *non-query* e comandos que retornam um conjunto de resultados.
- **Classe *ConnectionCE*:** Representa uma conexão com um bancos de dados *SQL Server Compact Edition*. Cria objetos da classe *CommandCE* retornado eles como um *Icommand*. É uma classe que pode ser usada de forma *singleton*.
- ***Generics* em C#:** Ajuda a definir classes que tem um comportamento comum, onde a diferença esta no tipo de dado manipulado. Exemplos: *Collection*<>, *List*<>, etc. Atributos estáticos em *generics* são únicos entre as classe que herdam a classe *generic*.
- **Classe *PersistClass*:**
  - Classe implementa toda a lógica de acesso ao banco, resolução dos mapeamentos e armazenagem do cache de acesso ao propriedades das classes mapeadas.
  - É uma *generic* que deve ser herdado pela classe DAO que representa um registro de uma tabela do banco de dados.

- Um construtor estático faz a inicialização das *queries* e cache de acesso das propriedades da classe base.
- **Métodos Update, Insert e Remove:** Fazem o papel das respectivas *queries update*, *insert* e *delete*.
- **Método Persist:** Método especial que executa um Update, caso este falhe ele executa um Insert.
- **Método Materialize:** Recupera do banco de dados as informações as quais a classe mapeia.
- **Método Select:** Monta uma *query* onde todos as propriedades não-nulas do objeto são usadas na clausula *where* da *query*.
- **Método SelectCollection:** Similar ao método Select, mas ele retorna uma coleção de objetos da classe que satisfazem a clausula *where*.
- **Método CopiarAtributos:** Usando o cache de acesso a propriedades este método faz a copia de dados de um *ResultSet* para as propriedades do objeto fornecido.
- **Método GetRestricoes:** monta a clausula *where* da *query* e adiciona parâmetros necessários ao *CommandCE* do objeto se for preciso.

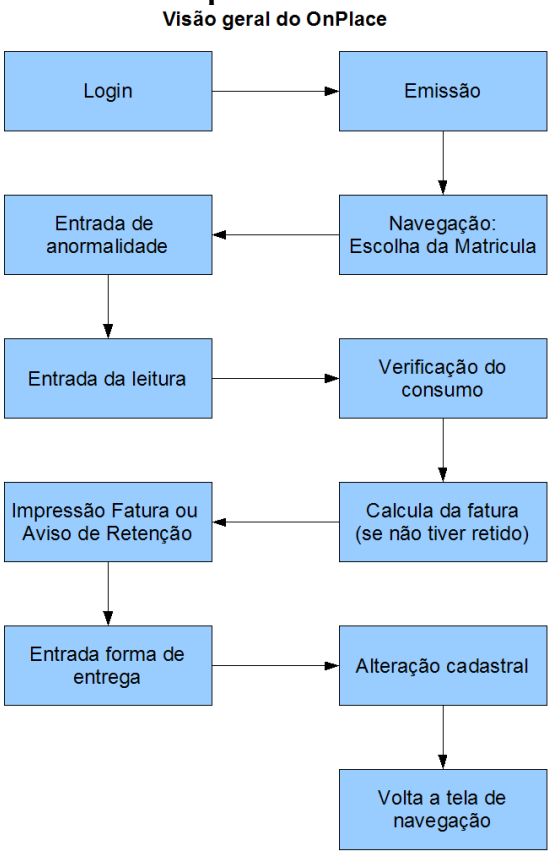
- **Classe *Log4CS*:** Fornece uma métodos simples de *log* para arquivo. Exitem vários níveis de *log*, conforme o nível selecionado atualmente mensagens de *log* podem ou não ir para o arquivo de *log*.

- **Classe *Singleton*:** Implementa o *design pattern singleton* na forma de um g*eneric*.

- **Classe *SystemTime*:** Utiliza *interop* para usar código não-gerenciado visando prover acesso a alteração da data do sistema.

OnPlace

- O projeto do *OnPlace* dependente do projeto *StrategosAPI*.
- O *OnPlace* utilizado pela SANED é um aplicativo feito em camadas. Cada aspecto do *OnPlace* tem um comportamento padrão que pode ter sido sobrescrito para satisfazer as necessidade da SANED.
- Todas as classes que foram personalizadas para a SANED estão em uma pasta chamada "Diadema" dentro do projeto do *OnPlace*.
- Usa "BuildTask.dll" para incrementar o número da versão a cada compilação.
- **Visão geral do funcionamento do aplicativo:**



- **Classe *ConfigXML*:** Classe contém vários métodos estáticos usados para ler o arquivo "config.xml". Este arquivo contem vários parâmetros do aplicativo.
- **Classes *ONP*:** Classes que representam tabelas do banco de dados começam com "Onp". Os mapeamentos são feito especificando o nome da tabelas e das colunas onde necessário. Estas classes contém métodos que são funcionalidades comuns que são usadas em várias partes do aplicativo. Classes *ONP* especificas da SANED:
  - Tradução de tipos entre o banco de dados e o C#: *val* para double, *dat* para Datetime,

ind/*des*/*nom*/*cod* para string e *seq* para int;
  - Em C# tipos de dados primitivos não aceitam null. Como os campo no banco de dados aceitam null e atribuir zero ou qualquer outro valor poderia causar problemas todos os mapeamentos feitos usam o *generic Nullable*. O atalho do compilador para este *generic* é coloca um sinal de interrogação como sufixo do tipo de dado. Exemplo: int? e double?. O único tipo de mapeamento que não precisa são as strings.
  - **Classe *OnpDescontoDiadema*:** Simples cadastro de descontos para matriculas.
  - **Classe *OnpFaturaImpostoDiadema*:** Contém os valores calculados de impostos para as contas que tem retenção de imposto na fonte.
  - **Classe *OnpImpostoDiadema*:** Cadastro simples de impostos que devem ser retidos na fonte.
  - **Classe *OnpMatriculaDiadema*:** Contém informações especificas de cada matricula em relação a SANED.

- **Classes Queries:** A idéia geral do aplicativo é não recorrer a executar *queries* no banco de dados para manter as "camadas" no aplicativo. Mas em alguns casos, principalmente na parte de cálculo de consumo e de conta, foram usadas algumas *queries* para acelerar os processos. As classes para estes casos são *QueriesCE* e *QueriesDiadema*.

- **Classes de teste:** Estas classes foram utilizadas para rodar testes em cima das classes de cálculo do OnPlace.
  - **Classes *ParametrosTeste*, *TstMatricula*, *TstMatriculaOcorrencia*:** São classes usadas para ler um arquivo XML através da deserialização do arquivo. Elas tem parâmetros de execução do teste como leitura, ocorrências e data na qual o teste deve ser executado.
  - **Classe *TesteControllerDiadema*:** Herda a classe *TesteController* que contém a lógica normal de uma entrada de ocorrência, leitura e emissão de conta ou de aviso de retenção. Esta classe foi personalizada para a SANED para lidar com os casos de leituras especiais.

- **Classes de impressão:** A pasta de "Impressao" contém classes que implementam a impressão na impressora "Zebra RW420".
  - **Classe *ZebraRw420*:** Implementa a interface *IPrinter* com métodos usados pela classe de impressão de conta.
  - **Classe *CodigoBarrasPadrao*:** Implementa a interface *ICodigoBarras*. Esta classe é herdade pela classe especifica da SANED, ela gera o código de barras e a linha digitável que são impressos na conta.

- **Telas (*Forms*):** Contém lógica para formatar e mostrar dados na tela. Estão sempre associadas a um classe controladora. As telas ficam em uma pasta chamada "Forms" dentro do projeto do *OnPlace*.

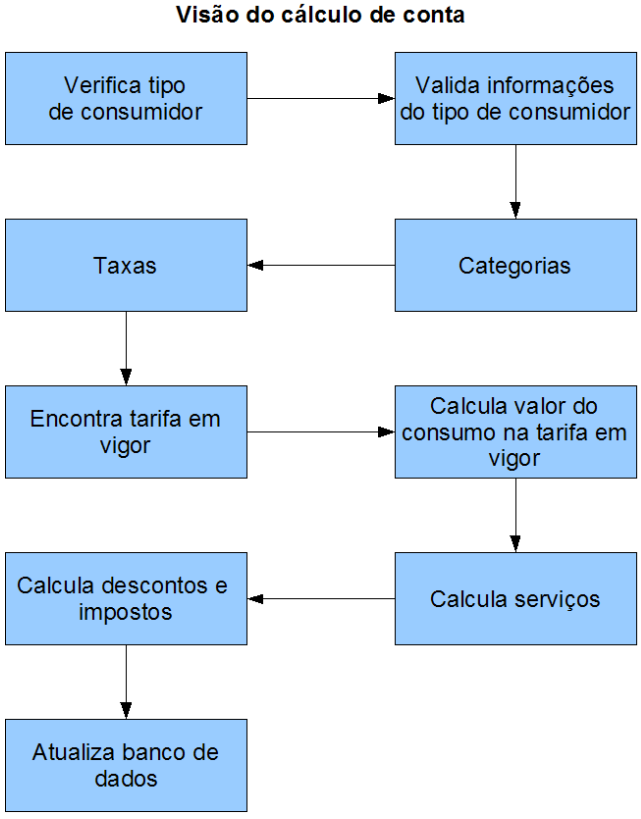
- **Classes controladoras:** São as classes que contém as regras de negócio. Podem ou não ter uma tela correspondente. As classes com comportante padrão ficam na pasta "Controladores" dentro do projeto do *OnPlace*.
  - **Classes principais:** *NavegacaoController*, *OcorrenciaController*, *LeituraController*, *CalculoTaxas*, *FormaEntregaController*, *ImpressaoDiadema*, *ImpressaoController* e *CalculoConsumo*.
  - **Classes secundárias:** *LoginController*, *MenuController*, *PesquisaController*,

BDController, *OpcoesNavegacaoController*, *ProgressoController*, *ServicosController*, *AlteracaoCadastralController*, *UpdateController*, *HistoricoConsumoController* e *ErrorController*.

- **Classe *ImpressaoDiadema*:** Esta é a classe que faz a impressão da conta, aviso de retenção, aviso de débito e teste de impressão.

- **Classe *CalculoTaxasDiadema*:** Herda a classe *CalculoTaxasPadrao* reimplementando alguns métodos para cumprir as regras da SANED.
  - O progresso do calculo é mostrado em uma tela.
  - Com base no tipo de consumidor da matricula executa regras especificas e depois, na maioria dos casos, segue o fluxo de cálculo mostrado abaixo. Quando o fluxo normal não é seguido é porque não foram lido todos os filhos da matricula ou porque a matricula não deve ser calculada (indicativo no banco de dados).

- **Visão geral do calculo de conta:**



Configurações

- Para desenvolvimento deve-se instalar os seguintes itens:
  - Para o *OnPlaceLoader* funcionar o "*Microsoft SQL Server 2005 Compact Edition*" deve ser instalado no computador: (~2MB)
    - http://www.microsoft.com/downloads/details.aspx?familyid=85e0c3ce-3fa1-453a-8ce9-af6ca20946c3&displaylang=en
  - Atualizar o *Visual Studio 2005* para última versão. Baixar um *Service Pack 1* disponível abaixo: (~430MB)
    - http://www.microsoft.com/downloads/details.aspx?familyid=bb4a75ab-e2d4-4c96-b39d-37baf6b5b1dc&displaylang=en
  - Instalar o *Service Pack 1* do .NET CF v2.0 no computador e no coletor: (~37MB)
    - http://www.microsoft.com/downloads/details.aspx?familyid=0c1b0a88-59e2-4eba-a70e-4cd851c5fcc4&displaylang=en
  - *Microsoft ActiveSync* 4.5: (~7MB, não é necessário instalar no *Windows Vista*)
    - http://www.microsoft.com/downloads/details.aspx?displaylang=pt-br&FamilyID=9E641C34-6F7F-404D-A04B-DC09F8141141
  - *Windows Mobile 6 SDK*: (~450MB, dependendo do que estiver instalado o tamanho da instalação pode ser maior)
    - http://www.microsoft.com/downloads/details.aspx?familyid=06111A3A-A651-4745-88EF-3D48091A390B&displaylang=en

- **Para copiar arquivo para o dispositivo:** No "Meu computador" existe um item chamado "Dispositivo móvel" ou "*Mobile Device*". Ao abrir este item você verá o "*My documents*" que esta dentro do emulador/dispositivo. Clicando em "Meu Dispositivo Baseado no *Windows Mobile*" você verá a pasta raiz do emulador/dispositivo.

- **Para conectar o emulador no ActiveSync:** Ir no ActiveSync, menu Arquivo, item "Configurações de conexão". Na janela que abre marcar a opção "Permitir conexões com um dos seguintes itens" e selecionar "DMA" na lista. Agora no menu *Tools* do *Visual Studio* existe uma opção "*Device Emulator Manager*". Ao clicar neste item um programa será rodado, nele esta listado todos os emuladores instalados na computador. Clicando com o botão direito do *mouse* sobre o emulador que esta sendo usado (emuladores em execução têm um ícone) aparece as opções para conectar o emulador no ActiveSync quando se clica em "Cradle". Para desconectar deve-se clicar no item "Uncradle". Para copiar arquivo para o emulador depois de clicar em "Cradle" é só seguir o processo normal de copia descrito acima.

- Na primeira execução, seja no dispositivo ou no emulador, o *Visual Studio* deverá instalar o SqlServer CE e a última versão do .NET CF. Caso isto não aconteça:
  - Copiar da pasta "C:\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0\wce500\armv4i" os arquivos "sqlce30.wce5.armv4i.CAB", "sqlce30.repl.wce5.armv4i.CAB" e "sqlce30.dev.ENU.wce5.armv4i.CAB" para o dispositivo/emulador e executar eles de dentro do dispositivo/emulador.
  - Por padrão o ".NET CF v2.0" já vem instalado no "*Windows Mobile 6*", mas pode ser necessário atualizar a *framework* para a versão "*Service Pack 1*". Copiar o arquivo

"NETCFv1.WM.ARMV4I.CAB" da pasta "C:\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\CompactFramework\2.0\v1.0\WindowsCE\WCE500\ARMV4i" e roda-lo no dispositivo/emulador.

- Copiar o arquivo "System.Data.SqlServerCe.dll" da pasta "C:\Program Files\Microsoft Visual Studio 8\SmartDevices\SDK\SQL Server\Mobile\v3.0" para a pasta do executável do *OnPlace* no dispositivo/emulador.

- O projeto ao qual se deseja rodar deve estar selecionado como "*StartUp Project*" no *Visual Studio*. Para isso clique bom o botão direito no projeto (*Solution Explorer*) e selecione "*Set as StartUp Project*".

- Para rodar o *OnPlace* em um coletor:
  - Selecionar "*Windows Mobile 6 Professional Device*" na barra de ferramentas do *Visual Studio* ou nas propriedades do projeto.
  - Para utilizar um dispositivo deve conectar a base com o dispositivo ao computador e o *ActiveSync* vai ser aberto. Seguir os passos na tela caso queira evitar que esta janela de configuração não se abra toda vez que o dispositivo for conectado.

- Para rodar o *OnPlace* no emulador:
  - Selecionar "*Windows Mobile 6 Classic Emulator*" ou "*Windows Mobile 6 Professional Emulator*" na barra de ferramentas do *Visual Studio* ou nas propriedades do projeto.
  - Para simular a conexão do emulador no computador deve usar o "*Device Emulator manager*". Ele se encontra disponível no menu *Tools* do *Visual Studio*.

- Alguns casos pode ser necessário dar um *reset* no dispositivo, para isso pressionar e segurar *Fn Azul* e *Enter* por pelo menos 6 segundos. O conteúdo da memória não é perdido.