1 Explicação de ficheiros e métodos

1.1 AppLogin.xaml.cs

- Responsável pela inicialização da aplicação;
- Responsável pela autenticação na aplicação.

- **Inicialização (AppLogin())**

  - Inicializa a cultura (linguagem) da aplicação através do que está gravado no ficheiro "Settings" Properties.Settings.Default.DefaultCulture.
  - Adiciona a password à conection string de forma a adicionar a password de encriptação da base de dados de login através do método – QueriesTableAdapter();
  - Inicializa a UI;

- **ButtonLogin_Click()**

  - Este método é chamado quando o botão de login é pressionado;
  - Armazena localmente o username e a password introduzidas;
  - Cria uma instância de SqliteLoginDataSet e preenche com os dados da base de dados.
  - Para cada linha da base de dados verifica se encontra um conjunto de Username e Password válidos.
  - Se não encontrar correspondência apresenta uma mensagem e fecha a aplicação.
  - Caso o login tenha sucesso:
    - colocar o atributo loginsuccessfull a true;
    - Armazenar a cultura em string na variável "locale" utilizando o método sanitize_locale(this);
    - Cria uma nova instância do FileManager, passando-lhe como argumento a cultura e o nível de permissões do utilizador que fez login;
    - Apresenta a nova janela de FileManager e fecha a AppLogin;

- **sanitize_locale(AppLogin appLogin)**

- Retorna a string relativa à cultura com base na linguagem escolhida na UI;

- **RadioLocalePT_Click(object sender, RoutedEventArgs e)**

  - É despoletado pela click na imagem PT;
  - Responsável por atualizar a cultura para Português no ficheiro settings e gravar;
  - Cria uma nova instância do AppLogin (que irá utilizar a linguagem alterada) e fecha a instância atual;

- **RadioLocaleEN_Click(object sender, RoutedEventArgs e)**

  - É despoletado pela click na imagem EN;
  - Responsável por atualizar a cultura para Inglês no ficheiro settings e gravar;
  - Cria uma nova instância do AppLogin (que irá utilizar a linguagem alterada) e fecha a instância atual;

- **MetroWindow_Loaded(object sender, RoutedEventArgs e)**

- Evento despoletado após a window ser carregada;
- Mediante a cultura atual, selecciona a bandeira correspondente;

- **ButtonCancel_Click(object sender, RoutedEventArgs e)**

  - Evento do botão de cancelar;
  - Desliga a aplicação;

- **TitleBarHelpButton_Click(object sender, RoutedEventArgs e)**

  - Abre o flyout com a ajuda relativa a esta window;

- **TitleBarSettingsButton_Click(object sender, RoutedEventArgs e)**

  - Botão para abrir uma nova window de alteração de password;

  - obtem o nome de utilizador, e a cultura e envia como parametro para uma nova instancia de PasswordChange();

- **QueriesTableAdapter(string connectionString)**

  - Adiciona a password de encriptação à connection string;

1.2 PasswordChange.xaml.cs

- **PasswordChange(string locale, string user, AppLogin parent)**

  - Recebe a cultura o user e a window em que a instância é criada;

- **ButtonChange_Click(object sender, RoutedEventArgs e)**

  - Obtém a senha, a nova senha e a confirmação da mesma;
  - Procura nos registos da BD a correspondência entre user e password;
  - Se encontrar correspondência, o utilizador está autencticado, e caso a nova senha e a confirmação de nova senha correspondam, atualiza a senha para a nova senha introduzida.

1.3 FileManager.xaml.cs

- **FileManager(string ChoosenLocale, int UserRole, MainWindow mainWindow)**

  - Recebe como argumento a cultura, o tipo de utilizador (permissões de utilizador) e a window principal (utilizado para o caso de ser instanciado na basewindow);

  - Lê a versão de software, para criar o caminho para a pasta de ficheiros de configuração.

  - Adiciona todos os ficheiros do tipo .prdgy à lista de ficheiros de configuração;

- **ListView_MouseDoubleClick(object sender, MouseButtonEventArgs e)**

  - Obtem o ficheiro em que foi feito o duplo clique e inicia uma nova MainWindow (página principal)

- **ListViewClick(object sender, SelectionChangedEventArgs e)**

  - Obtem o ficheiro que foi clicado e acede à base de dados correspondente de forma a obter os dados introduzidos na tabela cliente de forma a dar uma preview da configuração ao utilizador;
  - Apresenta na UI os dados lidos da DB;

- **NewConfigTile_Click(object sender, RoutedEventArgs e)**

  - Cria uma nova janela para criar um novo ficheiro;

- **DeleteConfigTile_Click(object sender, RoutedEventArgs e)**

  - Move o ficheiro de configuração selecionado para uma pasta 'old' que se encontra na pasta de ficheiros de configuração; Acaba por 'apagar' do sistema, sem se perder o ficheiro.
  - Caso o ficheiro já exista com o mesmo nome na pasta 'old', substitui pelo ficheiro apagado mais recentemente.

1.4 CreateNewFile.xaml.cs

- **CreateNewFile(ItemCollection Fileitems, string locale, int role, object fileManager)**

  - Recebe como argumentos:
    - Coleção com os ficheiros existentes na pasta de configurações; (utilizado para criar um novo ficheiro baseado num existente)
    - cultura do sistema;
    - permissões do utilizador;
    - O FileManager que o instanciou;

  - Preenche a combobox com os ficheiros existentes;

- **ButtonCreateNewFile_Click(object sender, RoutedEventArgs e)**

  - Evento do botão de criar um novo ficheiro;

  - Verifica se o nome do novo ficheiro está preenchido. Se estiver vazio retorna erro;

  - Verifica se o nome da configuração já existe. Caso já exista retorna erro;

  - Verifica o radio button que se encontra seleccionado.
    - Caso seja uma nova configuração baseada no ficheiro default copia o ficheiro default que se encontra em "@"\configuration\setup\default\default.prgy"". Caso este ficheiro não esteja disponivel, vai buscar um ficheiro redundante que se encontra em "Environment.GetFolderPath(Environment.SpecialFolder.SystemX86) + @"\default.prgy"";
    - Caso a nova configuração seja criada com base numa configuração existente, é copiado o ficheiro selecionado na combobox com um novo nome para a pasta de configuraçãoes;

  - Caso seja uma configuração baseada no ficheiro default inicia o wizard(WizardSystemArchitecture). Se não for, inicia a window principal (Mainwindow);

1.5 DataChoose.xaml.cs

- **DataChoose(MainWindow window_parent, bool action_type, string pre_selected_structure)**
  - Usado para escolher os dados a serem enviados/lidos para a central;
  - Recebe como argumento a window que o instancia, o tipo de acção (upload ou download) e a estrutura pré selecionada (no caso de ser efetuada uma leitura/escrita individual (botões individuais dentro de cada estrutura);
  - Caso a estrutura pré-selecionada seja uma string vazia “”, então todas as checkboxes estão a 'true';
  - Caso o parametro 'pre_selected_structure' seja diferente de uma string vazia, então vai seleccionar a checkbox correspondente à string passada como argumento.

- **ButtonReadOrWrite_Click(object sender, RoutedEventArgs e)**
  - Evento de click no botão de leitura/escrita; Armazena o valor das checkboxes;
  - Caso esta window seja aberta em contexto de leitura da central, invoca o método RequestDataFromProdigy(CheckboxesValues);
  - Caso seja aberta em contexto de escrita, invoca o método SendDataToProdigy(CheckboxesValues);

1.6 Settings.xaml.cs

- **Settings(MainWindow parent_window)**
  - Ao ser instanciado, verifica a cultura e força a bandeira correspondente na UI;

- **RadioLocalePT_Click(object sender, RoutedEventArgs e)**
  - É despoletado pela click na imagem PT;
    - Responsável por atualizar a cultura para Português no ficheiro settings e gravar;
    - Verifica se há alterações de dados e caso haja, pergunta ao utilizador se pretende gravar antes de reiniciar, decastar as alterações ou cancelar as alterações.
    - Cria uma nova instancia da window principal (BaseWindow) com a nova cultura;

- **RadioLocaleEN_Click(object sender, RoutedEventArgs e)**
  - Semelhante ao método RadioLocalePT_Click mas para a língua Inglesa.

1.7 RealTimeActions.xaml.cs

- **RealTimeActions(MainWindow mainWindow, Tile clicked_tile, byte MessageType)**
  - UI para efetuar ações na tab de Real Time; utilizado para ações de zonas, partições e saídas;
  - Recebe como argumentos, o objecto Tile que foi clicado, e o código de mensagem (código que distingue se a mensagem é para zonas, partições ou saídas);
  - Apresenta os botões de acção correspondente, mediante a vartiável MessageType;
- **SendAction(object sender, RoutedEventArgs e)**
  - Evento associado a todos os botões da UI;
  - Cria uma nova instância de RealTimeUserCodeInput() para se colocar o código de utilizador;

1.8 RealTimeUserCodeInput.xaml.cs

- **RealTimeUserCodeInput(RealTimeActions previous_menu, int object_id, MainWindow mainWindow, Button button_clicked, byte MessageType)**
  - Recebe como argumentos o 'object_id' que significa o número (da zona, ou partição ou saída), o botão que foi clicado e o tipo de mensagem.
  - Obtem a acção através da tag do botão de acção recebido por argumento.
- **ButtonSendUserCode_Click(object sender, RoutedEventArgs e)**
  - Obtem o código introduzido na textbox;
  - Se o código introduzido tiver menos do que 8 digitos, acrescenta-se 0xFF aos dígitos não introduzidos. É necessário garantir sempre um código de 8 dígitos.
  - Utiliza o método send_action(id, action, code, mainWindow, MessageType);
  - Envia como argumentos o 'id' (número da zona, saída ou partição), 'action' (código da acção – p.exe armar ou desarmar), código de utilizador num array de 8 bytes, a mainwindow e 'MessageType'(número que distingue se é uma ação para partições, zonas ou saídas);

1.9 WizardSystemArchitecture.xaml.cs

- **WizardSystemArchitecture(string locale, int role, string config_file_name)**
  - Recebe como argumentos a cultura, o grau de permissões do utilizador, e o nome do ficheiro criado;
- **ButtonCancel_Click(object sender, RoutedEventArgs e)**
  - Cancela o wizard, inicia a window principal da aplicação sem as configurações definidas no wizard;
- **ButtonOk_Click(object sender, RoutedEventArgs e)**
  - Cria um dicionário para cada uma das configurações (Keypad, Dialer, Partitions, Phones) em que a key é o 'id' e o value indica se foi selecionado (p. exe. SelectedPhone1.IsVisible);

1.10 Protocol/General – Protocolo de nível baixopara envio de mensagens para a central

- Esta classe implementa o nível baixo do protocolo de comunicação. Todas as classes que se encontram dentro da pasta Protocol utilizam métodos desta classe;
- **Método check_ID(MainWindow current_form):**
  - Envia a primeira mensagem para a central, após o connect;
- **Método update_hour_and_date(MainWindow current_form):**
  - Este método atualiza a hora e data do sistema, utilizando a hora e data do PC;
- **Método calculate_checksum(byte[] buf);**
  - Calcula o checksum para as mensagens a serem enviadas para a central;
- **Método send_msg(uint size, byte[] buf, byte[] cp_id, MainWindow current_form)**
  - Método utilizado para enviar todas as mensagens para acentral; (pedidos de leitura, pedidos de escrita, etc);
  - Recebe o buffer com os dados a enviar (buffer preparado nas outras classes dentro de Protocol), e o tamanho do buffer;
  - Aqui é adicionado o Header, o Id da central, o tamanho dos dados e checksum, ou seja é adicionado os blocos necessários para os dados se tornarem numa trama válida para a central;

1.11 Protocolo/StateMachine – Máquina de estados para interpretação das mensagens da central

- Nesta classe, são utilizados métodos para garantir a validade das tramas recebidas pela central e armazenar os dados para mais tarde serem processados;

- **ValidateData(byte data)**
  - Máquina de estados para validação das tramas recebidas;
  - À medida que cada byte é recebido, vai-se avançado no estado da máquina de estados;
  - No estado 1 e 2 valida-se o header;
  - No estado 3 armazena-se o tamanho dos dados;
  - No estado 4 armazena os dados a serem utilizados da trama que recebe, e o checksum para que possa ser validado;

- **ValidateReceivedChecksum(byte[] checksum)**
  - Compara o checksum recebido com o checksum calculado, caso sejam iguais retorna 'true', se não retorna 'false';

- **calculate_checksum(byte[] buf)**
  - Calcula o checksum através da tabela de checksum;

- **Operations(byte action, MainWindow mainform, int data_size)**
  - Faz um 'switch – case' com o byte corrrespondente à acção (escrever, ler, etc) e faz o processamento mediante o caso;
  - Este método é responsável por chamar os métodos responsáveis por atualização de dados no dataset ou UI;

2 Explicação de desenvolvimento – tarefas comuns

2.1 Como criar um novo ficheiro default

- Garantir que a central tem as configurações de fábrica;
- Caso não tenha, entrar com o código (77777777) no teclado, ir a sistema e ligar o bit 6 nas opções 3; Sair dos menus e esperar que a central reinicie;
- Após a central estar com as configurações por defeito, criar um novo ficheiro de configuração na aplicação.
- Ler a configuração da central para o PC;
- Guardar a configuração (CTRl+S ou ir a ficheiro → guardar);
- Copiar o ficheiro de configuração criado (Localiza-se nos Documentos do utilizador/Sanco S.A./Prodigy Configurator/ Vx.x.x/);
- Colar na pasta de desenvolvimento (ProdigyConfigToolWPF\configuration\setup\default\) com o nome default.prdgy
- A partir deste ponto todas as novas configurações serão criadas com base nesta nova configuração existente;

Casos de Uso:

- Alteração dos valores default da central (no código do Fernando), obriga à criação de um novo ficheiro default na aplicação de PC;

2.2 Como encriptar/desencriptar as Base de dados da APP

- Esta encripação é feita através de password;
- Para encriptar e desencriptar por password é utilizado o programa SQLITER que se encontra adicionado à solução;
- A password de encriptação é 'idsancoprodigy2017' para ambas as base de dados, a de login e as de configuração;
- É necessário introduzir o nome de uma tabela da BD no software SQLiter, para o caso da default.prdgy pode ser “User”, para o caso da DB de login 'SqliteLogin.prdgy' deve ser utilizada a tabela “UserLogin”;
- A password deve ser preenchida antes de seleccionar o ficheiro sqlite correspondente, se esta estiver encriptada;
- Para adicionar a password, após aparecer a label 'connected' por baixo do botão 'Browse', carregar no botão 'Set'. Para remover a encriptação carregar no botão 'Remove';

2.3 Base de dados – explicação

- Na aplicação existem duas base de dados, a default.prgy e a SqliteLogin.prgy (nome a ser mudado);

- A default é responsável por armazenar configurações da central;

- A SqliteLogin é responsável por armazenar as autenticações dos utilizadores da aplicação windows;

Tabelas da DB SqliteLogin:

  - UserLogin:
    - Responsável pelo armazenamento das informações de autenticaão e permissões do utilizador na aplicação;

Tabelas da DB default:

  - Area: Contém as configurações das partições;

  - AudioCustomized: Armazena as informações do audio criado pelo utilizador;

  - AudioDefault: Armazena as informações do audio reservado do sistema (p.exe: Id; descrição; caminho para o ficheiro);

  - AudioSystemConfiguration: Armazena as configurações do áudio de sistema; Armazena as opções de faixa de aúdio escolhida para determinados eventos;

  - Dialer: Contém as configurações do comunicador;

  - Event: Armazena os eventos do sistema, lidos da central;

  - GlobalSystem: Contém as configurações gerais de sistema;

  - Keypad: Contém as configurações dos teclados;

  - MainInfo: Armazena as informações do cliente a que a configuração pertence;

  - Output: Contém as configurações das saídas;

  - Phone: Contém as configurações dos telefones;

  - Timezone: Contém as configurações dos horários;

  - User: Contém as configurações dos utilizadores;

  - Zone: Contém as configurações das zonas;

2.4 Adicionar um elemento a uma estrutura

- Uma implementação anterior que é semlhante ao que será descrito encontra-se aqui: https://bitbucket.org/sancord/prodigyconfigtoolwpf/pull-requests/56/add-improvements-to-dialer-structure/diff , que corresponde a esta story: https://trello.com/c/EoiAlzqL/183-add-to-dialer-structure-uint8-ringcountermax-at-the-end-of-dialer-structure

- Assumindo que é necessário adicionar uma variável 'teste' à estrutura do telefones (poderia ser outra qualquer), e sendo a variável indicada pelo Fernando da seguinte forma “UINT32 teste”;

- Esta variável é adicionada ao fim da estrutura, ou seja, é a última variável da estrutura indicada pelo Fernando:
  - P.exe:

```
{
    …...
    …...
    UINT8 minute_test;
    UINT8 week_days;
    UINT32 teste; <------

}phones_flash_t
```

- Existem diversas abordagens para adicionar este elemento à estrutura da aplicação. É possível começar por adicionar o elemento à base de dados e UI, ou começar pela processamento de nível mais baixo, que será a forma que irei descrever;

Adicionar atributo ao dicionário attributes da classe:

- Adicionar a variável teste ao dicionário 'attributes', que se encontra na class User (Protocol/Phone.cs); Uma vez que a variável teste foi adicionada ao fim da estrutura phones_flash_t da central, então também terá de ser a última da nossa estrutura;

- Uma vez que é um UINT32, ocupa 4 bytes de memória, portanto o valor da key “value”, será 'new byte[4]';

- Para definir o valor da key “address”, é preciso verificar o valor da variável anterior, neste caso, é o atributo “week_days” que tem como address o valor **106** , e o valor da key 'value' que é **'new byte[1]'**. Com esta informação é possível calcular o valor para o novo atributo 'teste'. Se o 'week_days' ocupa 1 byte em memória, e se encontra na posição 106 na estrutura, o espaço de memória seguinte que se encontra livre é igual a (106+1) = 107;

- A key “data_grid_view_addr” não é necessária. Fazia parte de uma implementação antiga, que já não é utilizada;

- Assim temos todos os dados para construir o atributo 'teste' no dicionário attributes da class Phone, da seguinte forma:

```
{
    "week_days",
    new Dictionary<string, object>
    {
        { "value", new byte[1] },
        { "address", 106 },
        { "data_grid_view_addr", 7 }
    }
},
{
    "teste",
    new Dictionary<string, object>
    {
        { "value", new byte[4] },
        { "address", 107 }
    }
}
```

Adicionar a coluna teste à tabela Phone (BD)

- Desencriptar a base de dados default.prgy (ver como **aqui**)

- Adicionar a nova coluna 'Teste' à tabela Phone;

Fazer um refresh ao dataset (defaultdataset)

- **Guardar a connection string:**

Ir às propriedades do projecto, clicar em settings e copiar o conteúdo da 'defaultConnectionString', que é atualmente –> attachdbfilename=|DataDirectory|\configuration\setup\default\default.prgy;data source=configuration\setup\default\default.prgy

- **Apagar dataset**

Apagar o ficheiro defaultDataSet.xsd;

- **Voltar a criar dataset**

Para efetuar este passo é necessário garantir que a base de dados está desencriptada.
Abrir a tab 'Data Sources' no visual studio, clicar em adicionar data source, seleccionar base de dados e criar uma nova conexão do tipo SqLite;
Seleccionar o ficheiro 'default.prgy' e testar a conexão.
Seleccionar todas as tabelas da base de dados e terminar o wizard.

- **Ajustar dataset p/ audio customizado**

Para já é necessário fazer um pequeno ajuste ao dataset na tabela de audios customizados. Clicar com duplo clique no ficheiro do dataset, seleccionar a coluna 'Id' da coluna AudioCustomized e ir às propriedades. Colocar o AutoIncrementSeed igual ao último index dos audios reservados + 1 e indicar o AutoIncrementStep como sendo 1;

- **Reajustar connection string**

Ir às propriedades do projecto, clicar em settings e copiar para 'defaultConnectionString', a connection string anteriormente guardada; Esta operação é necessária, uma vez que o wizard do dataset cria uma connection string com caminho absoluto, e é pretendido um caminho relativo;

Leitura da central

Introdução

- A leitura da central é despoletada pelo utilizador na window DataChoose através do método `ButtonReadOrWrite_Click(object sender, RoutedEventArgs e);`
- Neste método utilizado um método da Basewindow (porque esta é que contém o objeto de porta série); O método utilizado no caso da leitura é `RequestDataFromProdigy();`
- Neste caso específico o mecanismo de **pedido de leitura** dos Phones já se encontra desenvolvida, se não teria de ser adicionado aqui de forma semelhante ao que se encontra desenvolvido. (p.exe nos expansores, seria necessário criar estes mecanismos de novo);
- Após o envio do pedido de leitura, é necessário processar a resposta. Este processamento é efetuado no método `UpdateDataGridViews()` que se encontra na Basewindow;
- Neste método é efetuado o processamento das respostas de leitura de todas as estruturas do sistema, e o respectivo armazenamento dos dados processados no dataset;

Implementação

- Para adicionar a leitura do atributo 'teste' da estrutura Phone, neste método procuramos pela região Phones;

- Nesta região começamos por efetuar a leitura do atributo 'teste' do buffer de dados recebidos. Para tal utiliza-mos o seguinte código:

```
byte[] teste= (byte[])phone.attributes["teste"]["value"];
for (int i = (7 + (int)phone.attributes["teste"]["address"]), j = 0; i < (7 +
    (int)phone.attributes["teste"]["address"] + phone.Length); i++, j++)
      {
          teste[j] = buf[i];
      }
```

  - Neste pedaço de código, é criada uma variável do tipo byte array do tamanho registado no dicionário através da linha `byte[] teste=(byte[])user.attributes["teste"]["value"];`

  - Depois é preciso ler estes dados na posição correta do array de bytes da resposta da central. Para tal é necessário fazer um ciclo que começa no valor armazenado na key 'address' do attributo teste + 7 (Este valor é o numero de bytes iniciais da trama que não traduzem dados relevantes para este processamento), e acaba no address + 7 + o tamanho da variável; E assim a variável teste é armazenada byte a byte através da linha `teste[j] = buf[i];`

- Após esta fase é necessário fazer o processamento necessário para que estes 4 bytes se transformem num inteiro interpretável pelo um humano, e armazenar este valor processado na coluna 'teste' da tabela Phone do dataset;

- Este processamento e armazenamento no dataset pode ser efetuado da seguinte forma:

```
databaseDataSet.Phone.Rows[phone_to_read]["Teste"] = (teste[3] << 24) + (teste[2]
<< 16) + (teste[1] << 8) + teste[0];
```

- Após o processamento descrito, garanti-mos que o novo atributo da estrutura é lido com sucesso para o dataset. Para confirmar é necessário colocar a UI coerente com o trabalho que foi efetuado, ouy analisar em debug o valor armazenada no dataset e comparara com o lido no teclado da central;

User Interface

- Após ter a leitura dos dados recebidos implementada, podemos apresentar o valor armazenado no dataset na datagrid dos Telefones;
- Para tal no BaseWindow.xaml é preciso localizar a 'MainPhonesTab', e dentro desta Tab, a Datagrid '`phoneDataGrid';`
- Nesta datagrid é necessário adicionar um código que permita aceder à coluna Teste da tabela Phone do dataset; É necessário adicionar o código seguinte:

```
<DataGridTextColumn Binding="{Binding Teste}" Header="{x:Static p:Resources.Teste}"/>
```

- No binding é necessário indicar o nome da tabela, de forma a ligar o conteúdo da datagridtextcolumn ao dataset, e no header a tradução (que necessitaria de ser criada) do header da tabela da datagrid;

- Para verificar se a implementação se encontra como desejado, entrar na aplicação e fazer uma leitura dos telefones; Verificar o valor lido com o lido no teclado da central;

Escrita para a central

Introdução

- A escrita da central é despoletada pelo utilizador na window DataChoose através do método `ButtonReadOrWrite_Click(object sender, RoutedEventArgs e);`

- Neste método utilizado um método da Basewindow (porque esta é que contém o objeto de porta série); O método utilizado no caso da leitura é `SendDataToProdigy();`

- Neste caso específico o mecanismo de **pedido de escrita** dos Phones já se encontra desenvolvida, se não teria de ser adicionado aqui de forma semelhante ao que se encontra desenvolvido. (p.exe nos expansores, seria necessário criar estes mecanismos de novo);

- O processamento dos dados presentes no dataset, para um array de bytes que será enviado para a central é feito no método Write() da Class Phones;

Implementação

- A implementação da escrita deve ser efetuada no método Write(), da classe correspondente, neste caso, na classe Phones;

- O objectivo principal aqui, é traduzir/processar o conteúdo que se encontra no dataset (que é apresentado na datagrid). Para tal é preciso aceder ao dataset, traduzir o conteúdo em arrays de bytes e por fim colocar no buffer de envio na posição correta;

- Aqui é preciso entender que tipo de dados se armazenou na coluna teste da tabela Phone. Anteriormente não descrevi que tipo de dados a coluna possui, porque pode ser texto ou um inteiro. Neste caso é indiferente armazenar os dados como texto ou como um numero inteiro. Eu armazenaria como texto, e portanto a solução apresentada é para o caso de ser texto /string.

- Inicialmente obtém-se os dados do dataset e faz-se um Parse para UINT, de forma a ser possível mais tarde obter os bytes que formam o UINT correspondente.

```
uint teste = uint.Parse(
                mainForm.databaseDataSet.Phone.Rows[(int)phone_number]["Teste"].ToString());
```

- Na linha anterior armazena-se o resultado de um Parse para UINT do valor armazenado no dataset. (p.exe: se for “30”, passaria ao número 30);

- Agora é necessário converter o UINT para byte array e armazenar numa nova variável. Para tal é utilizado o método BitConverter.GetBytes(), da seguinte forma:

  ```
  byte[] teste_bytes = BitConverter.GetBytes(teste);
  ```

- Neste momento os dados encontram-se prontos para serem colocados no array de envio. Estes dados têm de ser colocados nas posições corretas do array, e para tal é necessário adicionar o seguinte código:

  ```
  for (i = ((int)attributes["teste"]["address"] + temp), j = 0; i < ((int)attributes["teste"]
  ["address"] + temp + (teste_bytes.Length)); i++, j++)
  {
    byte_array[i] = teste_bytes[j];
  }
  ```

- Para testar, escrever um valor na coluna teste da datagrid Phone e enviar para a central; Posteriormente, ler no teclado o valor e verificar se corresponde;

2.5 Criar uma estrutura nova

- A criação de uma estrutura de forma semelhante ao colocado aqui, encontra-se aqui : https://bitbucket.org/sancord/prodigyconfigtoolwpf/pull-requests/11/implement-timezone-class-and-dataset/diff

- Vamos assumir que é necessário criar a estrutura de Expansores - “Expanders”

Criar uma classe nova

- Criar uma nova classe dentro da pasta Protocol com o nome Expander;

Criar dicionário de attributes

- Adicionar o dicionário attributes à semelhança do que existe nas outras classes. Os atributos que se deve adicionar ao dicionário, vêm da estrutura que o Fernando disponibiliza, à semelhança do descrito aqui;

Criar tabela nova e preencher com colunas

- É necessário adicionar uma nova tablea ao default.prdgy.
- Para adicionar a tabela, é preciso desencriptar a tabela (ver **aqui**).
- Adicionar a tabela com as colunas em correspondência aos atributos do dicionário;

Fazer refresh ao dataset

- Após alterar a tabela, é necessário apagar e voltar a criar o dataset (ver **aqui**);

Adicionar referência do dataset à UI

- Adicionar referência na vista, ao dataset da seguinte forma no BaseWindow.xaml

```
<CollectionViewSource x:Key="expanderViewSource" Source="{Binding Expander,
                                                    Source={StaticResource databaseDataSet}}"/>
```

Preencher o dataset no arranque

- É necessário preencher o dataset no arranque da BaseWindow.
- É preciso que os dados de cada tabela da BD sejam passados para cada tabela do dataset;
- O código seguinte representa uma forma de o fazer para os expansores no evento de baseWindow loaded:

```
        ExpanderTableAdapter databaseDataSetExpanderTableAdapter = new
ExpanderTableAdapter();
        databaseDataSetExpanderTableAdapter.Fill(databaseDataSet.Expander);
        System.Windows.Data.CollectionViewSource expanderViewSource =
((System.Windows.Data.CollectionViewSource)(this.FindResource("expanderViewSource")));
        expanderViewSource.View.MoveCurrentToFirst();
```

Criar UI (Nova tab)

- Adicionar uma nova tab ao Tabcontrol (à semelhança das Tabs existentes)
- Adicionar a datagrid à semelhança das outras existentes no projecto;
- Adicionar as colunas à datagrid e fazer o binding ao nome das colunas da tabela;(ver **aqui**)

Criar mecanismos de leitura e escrita novos

- Criar os métodos de read e de write, à semelhança das outras estruturas existentes;
- Processar a leitura de forma semelhante ao que é descrito aqui, mas para todos os atributos desta nova estrutura;
- Processar a escrita de forma semelhante ao que é descrito aqui, mas para todos os atributos desta nova estrutura;

2.6 Como gravar os dados introduzidos no dataset para a base de dados

- Para gravar os dados alterados no sistema, ou seja, alterados nas datagrids, é possível utilizar o atalho de CTRL + S ou clicar em ficheiro + gravar;

- Todas as estruturas de dados são armazenadas apenas quando o utilizador decidir que deve gravar, excepto na leitura de eventos do sistema. Neste caso os eventos são imediatamente gravados na base de dados;

- O método responsável por armazenar os valores do dataset na base de dados é o método Save_Database_data();

- No caso dos eventos, este Save para a base de dados é feito no método UpdateDataGridViews() na região em que se encontram os eventos atrvés do código:

```
    databaseDataSet.Event.Rows.Add(new_event);
EventTableAdapter databaseDataSetPhoneTableAdapter = new EventTableAdapter();
databaseDataSetPhoneTableAdapter.Update(databaseDataSet.Event);
```