



LABORATÓRIO NACIONAL
DE ENGENHARIA CIVIL

SOFTWARE DT500COM

Controlo dos Geologgers da Série DT500, da Datataker

Lisboa • outubro de 2016

I&D ESTRUTURAS

RELATÓRIO 286/2016 – DE/NOE

Título

SOFTWARE DT500COM

Controlo dos Geologgers da Série DT500, da Datataker

Autoria

DEPARTAMENTO DE ESTRUTURAS

Paulo Silveira

Investigador Principal, Núcleo de Observação de Estruturas

CONSELHO DIRETIVO

José Barateiro

Investigador Auxiliar, Núcleo de Tecnologias da Informação em Engenharia Civil

Copyright © LABORATÓRIO NACIONAL DE ENGENHARIA CIVIL, I. P.

AV DO BRASIL 101 • 1700-066 LISBOA

e-mail: lnec@lnec.pt

www.lnec.pt

Relatório 286/2016

Proc. 0304/3212/03043212

SOFTWARE DT500COM

Controlo dos Geologgers da Série DT500, da Datataker

Resumo

Neste relatório apresenta-se o *software* desenvolvido para controlo das subunidades de aquisição Geologger série 500, da Datataker.

Para além de se fornecerem indicações para trabalhar com o *software* referido, é feita uma apresentação do projeto desenvolvido na linguagem de programação C#.

Palavras-chave: Porta série / Data logger / C#

SOFTWARE DT500COM

Control of DataTaker's Geologgers from series DT500

Abstract

This report presents the software developed to control series 500 Geologgers, of Datataker.

Besides the information needed to work with this software, it is also presented the C# project developed.

Keywords: Serial port / Data logger / C#

Índice

1	Introdução	1
2	Instruções de utilização do software DT500Com	2
2.1	Form DT500	2
2.2	Abertura e fecho da porta série	4
2.3	Reset do <i>data logger</i>	4
2.4	Manipulação da <i>text box</i> inferior	4
2.5	Envio, arranque e paragem do programa para o <i>logger</i>	5
2.6	Criação de um ficheiro com as listagens dos programas	7
2.7	Gravação dos dados para ficheiro	8
2.8	Visualização das leituras em tempo real num gráfico	8
2.9	Testes ao <i>logger</i> e à informação armazenada	10
2.10	Operações relativas ao cartão de memória	12
2.11	Monitorização.....	12
2.12	Alterações ao programa destinadas à utilização na monitorização de estruturas	13
3	Comandos de programação do <i>logger</i>	15
3.1	Medição de temperaturas com termómetros do tipo PT100	16
3.2	Medição de rotações com clinómetros Schaevitz.....	16
3.3	Medição de extensómetros de corda vibrante.....	16
3.4	Medição de deslocamentos com defletómetro potenciométrico (I)	17
3.5	Medição de deslocamentos com defletómetro magnetostritivo (I)	17
3.6	Medição de deslocamentos com defletómetro magnetostritivo (V).....	17
3.7	Medição de pressão.....	18
3.8	Medição de extensões com extensómetros em ponte completa.....	18
4	Descrição do <i>software</i> DT500Com	19
4.1	Introdução a alguns conceitos da linguagem C#.....	19
4.1.1	Abstração	20
4.1.2	Encapsulamento	21
4.1.3	Herança.....	21
4.1.4	Polimorfismo	21
4.2	Código.....	22
4.2.1	Ficheiro Form1.cs	22
4.2.2	Ficheiro FormLimY.cs	23
4.2.3	Ficheiro FormMemoria.cs	24
4.2.4	Ficheiro FormRun.cs.....	24
4.2.5	Ficheiro FormSched.cs	24
4.2.6	Ficheiro FormStatus.cs	25
4.2.7	Ficheiro FormTest.cs	26
4.2.8	Ficheiro GestorHora.cs	26
4.2.9	Ficheiro GestorPortaSerie.cs.....	26
4.2.10	Ficheiro Preferencias.cs	27
4.2.11	Ficheiro SerialThread.cs	28
4.2.12	Ficheiro Util.cs	28
5	Conclusões.....	30
	Anexos.....	31
	ANEXO I Form1.cs	33

ANEXO II FormLimY.cs.....	51
ANEXO III FormMemoria.cs.....	55
ANEXO IV FormRun.cs.....	59
ANEXO V FormSched.cs.....	65
ANEXO VI FormStatus.cs.....	73
ANEXO VII FormTest.cs.....	77
ANEXO VIII FormUnload.cs.....	81
ANEXO IX GestHora.cs.....	87
ANEXO X GestPortaSerie.cs.....	93
ANEXO XI Preferences.cs.....	99
ANEXO XII Program.cs.....	103
ANEXO XIII SerialThread.cs.....	107
ANEXO XIV Util.cs.....	111

Índice de figuras

Figura 1.1 – Conversor USB/COM.....	1
Figura 2.1 – <i>Form</i> DT500.....	2
Figura 2.2 – Grupo de opções “Porta Série”.....	4
Figura 2.3 – Grupo de opções “Porta Série”.....	4
Figura 2.4 – <i>Botões</i> para limpeza ou cópia da <i>text box</i> inferior.....	5
Figura 2.5 – Envio do programa para o <i>logger</i>	5
Figura 2.6 – Escolha do ficheiro com o programa a enviar.....	6
Figura 2.7 – Arranque das <i>schedules</i>	6
Figura 2.8 – Paragem das <i>schedules</i>	7
Figura 2.9 – Listagens dos programas.....	7
Figura 2.10 – Gravação dos dados armazenados para um ficheiro.....	8
Figura 2.11 – Visualização dos dados em tempo real num gráfico.....	9
Figura 2.12 – Alteração da escala das ordenadas do gráfico.....	9
Figura 2.13 – Teste ao status do <i>logger</i>	10
Figura 2.14 – Teste ao <i>hardware</i> do <i>logger</i>	11
Figura 2.15 – Teste à memória do <i>logger</i> e do cartão.....	11
Figura 2.16 – Teste ao cartão de memória.....	12
Figura 2.17 – Operações relativas ao cartão de memória.....	12
Figura 2.18 – Monitorização.....	13
Figura 3.1 – Programa para ser enviado para a memória interna.....	15
Figura 3.2 – Programa para ser enviado para o cartão de memória.....	15
Figura 3.3 – Medição de temperaturas com termómetros do tipo PT100.....	16
Figura 3.4 – Medição de rotações com clinómetros Schaevitz.....	16
Figura 3.5 – Medição de extensómetros de corda vibrante.....	16
Figura 3.6 – Medição de deslocamentos com defletómetro potenciométrico (I).....	17
Figura 3.7 – Medição de deslocamentos com defletómetro magnetostritivo (I).....	17
Figura 3.8 – Medição de deslocamentos com defletómetro magnetostritivo (V).....	17
Figura 3.9 – Medição de pressão.....	18
Figura 3.10 – Medição de extensões com extensómetros elétricos de resistência em ponte completa.....	18
Figura 4.1 – Imagem do <i>Solution Explorer</i>	19

Índice de quadros

Quadro 4.1 – Métodos definidos em Form1.cs	22
Quadro 4.2 – Métodos definidos em FormLimY.cs	23
Quadro 4.3 – Métodos definidos em FormMemoria.cs	24
Quadro 4.4 – Métodos definidos em FormRun.cs	24
Quadro 4.5 – Métodos definidos em FormSched.cs	25
Quadro 4.6 – Métodos definidos em FormStatus.cs	25
Quadro 4.7 – Métodos definidos em FormTest.cs	26
Quadro 4.8 – Métodos definidos em GestorHora.cs	26
Quadro 4.9 – Métodos definidos em GestorPortaSerie.cs	26
Quadro 4.10 – Métodos definidos em Preferences.cs	27
Quadro 4.11 – Métodos definidos em SerialThread.cs	28
Quadro 4.12 – Métodos definidos em Util.cs	28

1 | Introdução

Neste relatório apresenta-se o *software* desenvolvido na linguagem de programação C# para controlar as subunidades de aquisição Geologger série 500, da Datataker.

O equipamento em causa não permite frequências de leitura superiores a 1 Hz, pelo que não é adequado à monitorização do comportamento dinâmico de estruturas. No entanto, a sua capacidade para funcionar com um vasto leque de transdutores, com princípios de funcionamento diferentes, torna-o extremamente versátil, sendo perfeitamente adequado para a realização de ensaios estáticos.

Dado que tem vindo a ser realizada a substituição destes equipamentos por subunidades de aquisição mais modernas, o seu reaproveitamento permite fazer, por exemplo, medição de temperaturas com custo reduzido.

O facto do equipamento se encontrar descontinuado, faz com que tenha sido abandonado o suporte ao *software* de controlo disponibilizado pelo fabricante, tendo por isso deixado de funcionar nos sistemas operativos mais recentes. Assim tornou-se premente desenvolver o *software* DT500Com para poder efetuar o reaproveitamento destes *data loggers*.

Para além deste capítulo introdutório, apresentam-se no Capítulo 2 | instruções para usar este *software*.

A programação dos *loggers* implica que o utilizador conheça as instruções necessárias para efetuar o seu comando. No Capítulo 3 | apresentam-se os comandos necessários para efetuar a medição com os tipos de sensores mais comuns. Estes comandos são escritos num ficheiro ASCII que é enviado para o *logger* pelo *software* DT500.

No Capítulo 4 | faz-se uma descrição do *software* desenvolvido e apresentam-se alguns dos conceitos utilizados na programação orientada por objetos.

Nos Anexos exibem-se as listagens dos principais ficheiros que constituem este projeto.

Dado que muitos computadores não possuem portas série para efetuar a ligação ao *logger* é necessário utilizar um conversor USB/COM (Figura 1.1). Chama-se, no entanto a atenção, para o facto destes conversores nem sempre possibilitarem utilizar a velocidade de comunicação mais alta permitida pelo *data logger* (9600 bits/s). É usual ter de reduzir esta velocidade para 4800 ou 2400 bits/s.



Figura 1.1 – Conversor USB/COM

2 | Instruções de utilização do software DT500Com

Ao iniciar-se o programa *DT500Com* surge uma *form* principal (DT500), a partir da qual é controlado o *logger*. Este *software* permite abrir a porta série desejada e seleccionar a velocidade de transferência adequada, assim como fechar a porta que foi aberta.

Estes *data loggers* necessitam que a porta série, a que se encontram ligados, possua a seguinte configuração:

- baud rate: 300, 1200, 2400, 4800, 9600 (configuração de fábrica);
- paridade: nenhuma;
- data bits: 8;
- stop bit: 1.

Para além do estabelecimento da comunicação, é possível enviar programas constituídos por conjuntos de instruções escritas em ficheiros ASCII, arrancar ou parar as *schedules* definidas, fazer testes ao *logger* e ao cartão de memória, fazer o *unload* dos dados armazenados no *logger* ou no cartão de memória e gerir a aquisição de dados criando ficheiros horários que contêm os valores das leituras efetuadas.

Esta aplicação tem a possibilidade de criar um ficheiro de configuração destinado a guardar a configuração da porta série e também definir se se pretende efetuar o seu arranque automático, de modo a facilitar a sua utilização na monitorização estrutural.

2.1 Form DT500

A *Form* “DT500” (Figura 2.1) é constituída por duas janelas de texto, sendo a superior reservada à escrita de comandos para o *logger* e a inferior destinada à visualização das respostas por este enviadas.

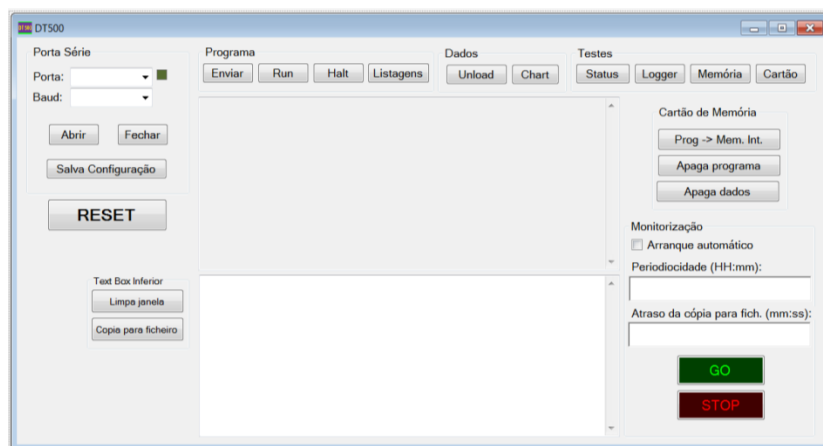


Figura 2.1 – Form DT500

A generalidade dos comandos enviados para o *logger* deve ser escrita em letra maiúscula. Excetua-se os *switches* que são definidos por um “/” seguido de uma letra que será maiúscula, no caso de ativação da função, ou minúscula caso contrário. Por exemplo “/E” permite ecoar os comandos para o computador, enquanto “/e” inibe que os comandos dados sejam ecoados para o computador.

Para além destas duas janelas existem sete grupos de opções e um botão “RESET” que provoca o *Reset* do *logger*. No grupo de opções “Porta Série” escolhe-se o baud rate e a porta, e faz-se a sua abertura e fecho. O botão “Salva Configuração” permite guardar a configuração no ficheiro *pref.xml*. Existe ainda uma luz que fica verde quando a porta se encontra aberta.

No grupo de opções “Programa” existem três botões. Um destina-se a desencadear o envio do programa para o *logger* e os outros dois a fazer o arranque ou a paragem das *schedules* definidas nesse programa.

No grupo de opções “Dados” é possível efetuar a gravação dos dados armazenados num ficheiro *.txt.

No grupo de opções “Testes”, encontram-se três botões. O botão “Status” possibilita conhecer:

- o endereço do *logger*;
- a versão da ROM;
- quais as *schedules* definidas e se estão ou não ativas;
- quantos alarmes se encontram ativos e inativos;
- a memória interna livre e ocupada;
- a memória do cartão livre e ocupada;
- a memória do programa livre e ocupada.

O botão “Logger” permite efetuar um teste ao hardware do *logger* e o botão “Memória” possibilita conhecer o nível de ocupação da memória interna e do cartão.

O grupo de opções “Cartão de Memória” possui três botões. O botão “Prog -> Mem. Int.” provoca a transferência do programa do cartão de memória para a memória interna do *logger*. Os botões “Apaga programa” e “Apaga dados” permitem, respetivamente, apagar o programa e os dados existentes no cartão de memória.

O grupo de opções “Text Box Inferior” possui dois botões. O botão “Limpa janela” desencadeia a limpeza do conteúdo da *text box* inferior e o botão “Copia para ficheiro” permite copiar o seu conteúdo para um ficheiro com extensão “.txt” e cujo nome se inicia por “TxBox” seguido da data e da hora.

No grupo de opções “Monitorização” existe uma *check box* que quando está marcada permite desencadear o arranque automático da aplicação. Existem ainda duas caixas de texto e dois botões. Nas caixas de texto indica-se a periodicidade da monitorização e o atraso com que se escreve, para um ficheiro, os valores correspondentes à última leitura. Os botões “GO” e “STOP” destinam-se a desencadear, respetivamente, o registo das leituras e a paragem desse registo para ficheiro.

2.2 Abertura e fecho da porta série

No grupo de opções “Porta Série” (Figura 2.2) faz-se a escolha de uma das portas disponíveis e também do baud rate da comunicação, cujo valor terá de ser igual ao configurado no *logger* por meio dos *dip switches* existentes para o efeito.

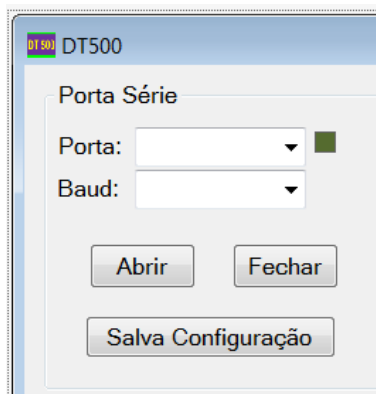


Figura 2.2 – Grupo de opções “Porta Série”

2.3 Reset do *data logger*

Sob o grupo de opções “Porta Série” (Figura 2.3) existe um botão de “RESET” que limpa os dados e o programa existente na memória interna. Se existir um programa no cartão de memória este botão faz arrancar esse programa. Os dados existentes no cartão de memória não são afetados.

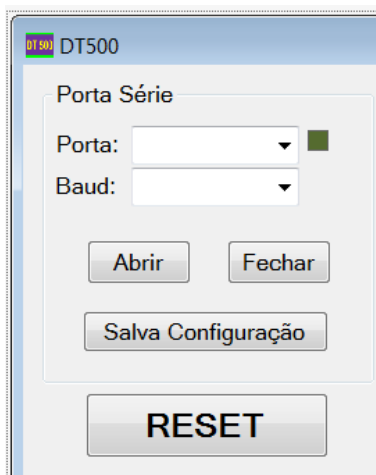


Figura 2.3 – Grupo de opções “Porta Série”

2.4 Manipulação da *text box* inferior

Sob o botão de “RESET”, na *group box* “Text box inferior” (Figura 2.4) existem dois botões. O botão “Limpa janela” desencadeia a limpeza do conteúdo da *text box* inferior. O botão “Copia para ficheiro” permite fazer a cópia do conteúdo da *text box* inferior para um ficheiro. Trata-se de um ficheiro ASCII, com o prefixo “TxBox” seguido da data e hora, e com extensão “.txt”.

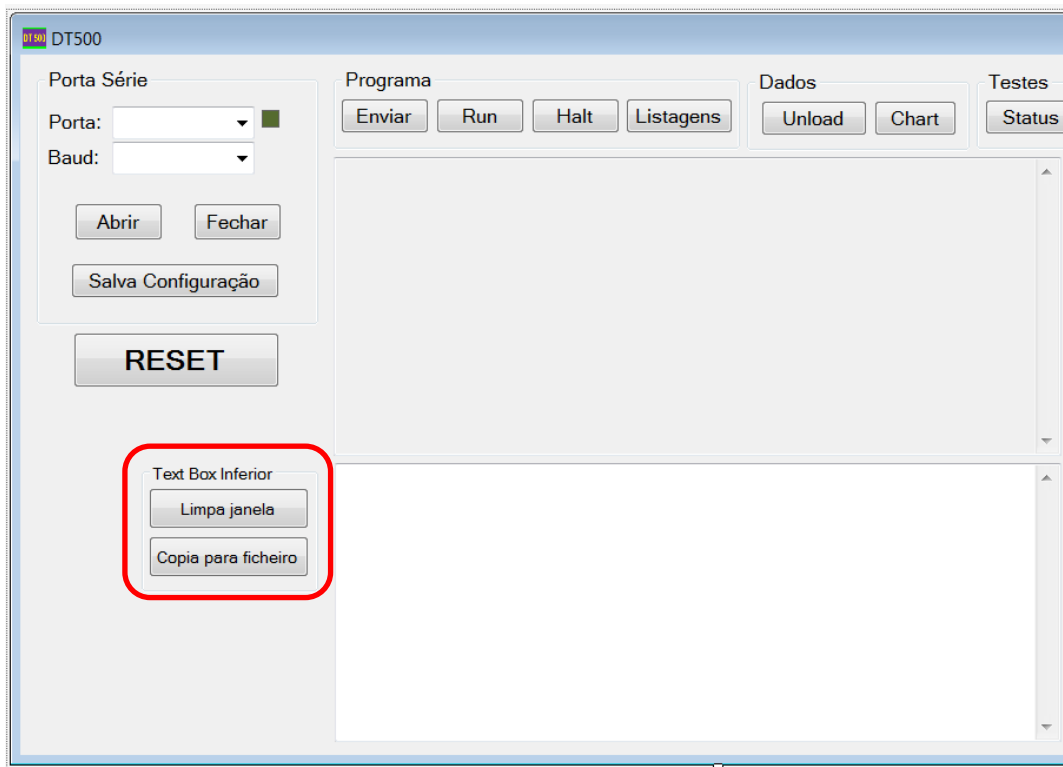


Figura 2.4 – Botões para limpeza ou cópia da text box inferior

2.5 Envio, arranque e paragem do programa para o *logger*

No grupo de opções “Programa” existem o botão “Enviar” (Figura 2.5) que desencadeia o envio das instruções guardadas num ficheiro a escolher pelo utilizador (Figura 2.6), o botão “Run” (Figura 2.7) que permite o arranque individual ou geral das *schedules* programadas e ainda o botão “Halt” (Figura 2.8) que possibilita a paragem individual ou geral das *schedules* programadas.

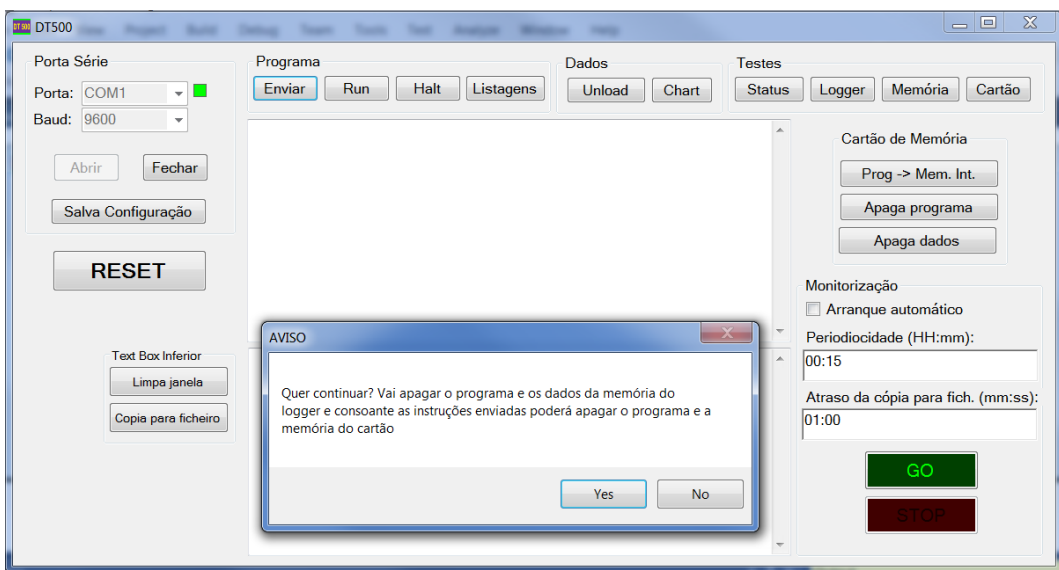


Figura 2.5 – Envio do programa para o *logger*

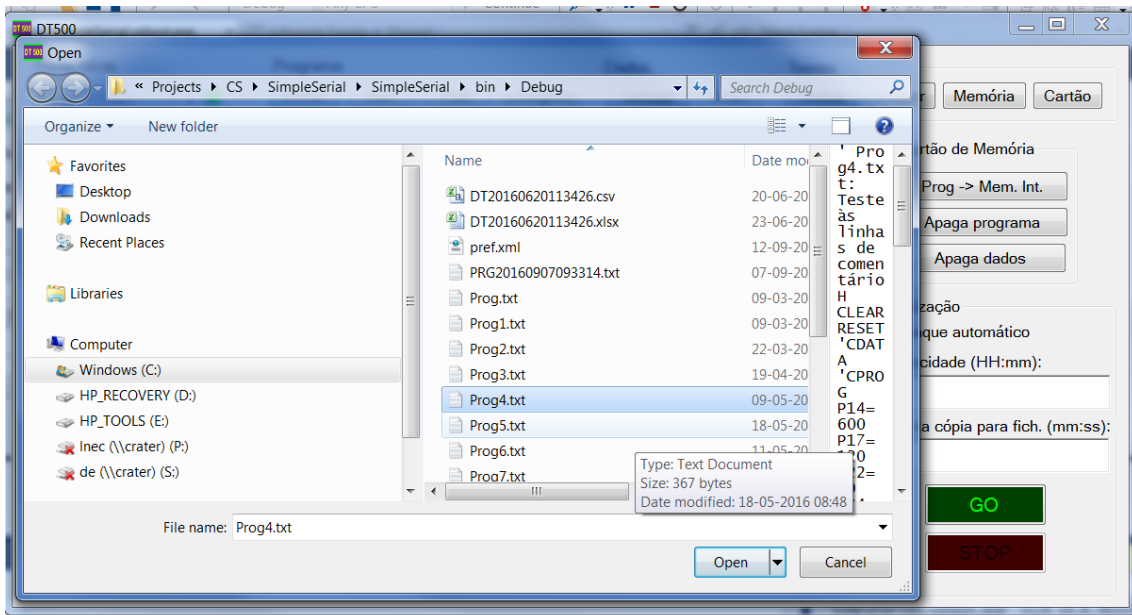


Figura 2.6 – Escolha do ficheiro com o programa a enviar

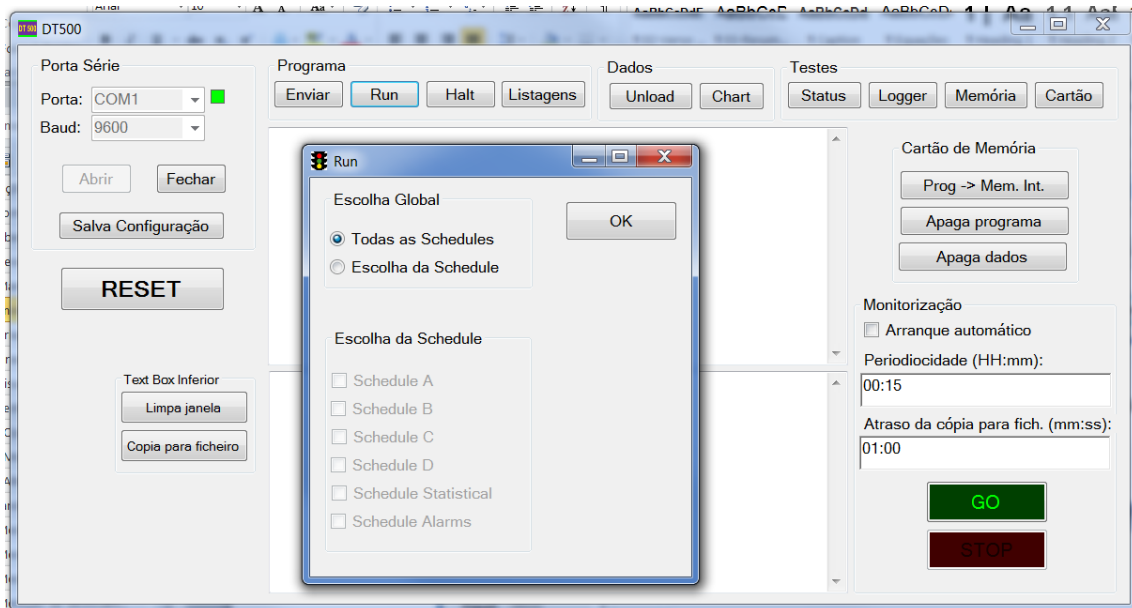


Figura 2.7 – Arranque das schedules

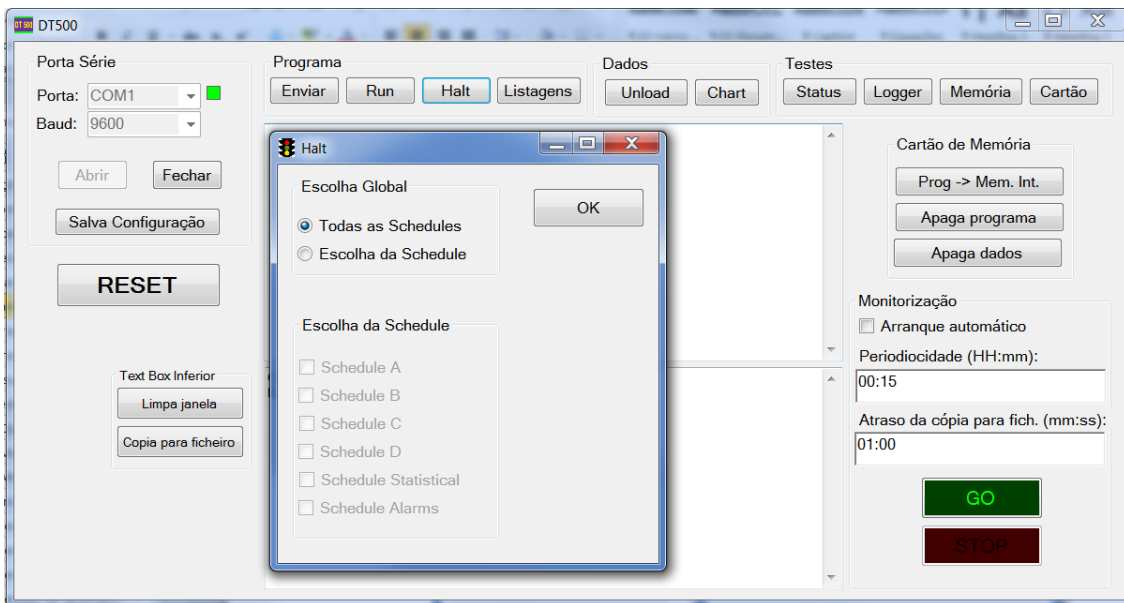


Figura 2.8 – Paragem das schedules

2.6 Criação de um ficheiro com as listagens dos programas

No grupo de opções “Programa” existem o botão “Listagens” (Figura 2.9) que permite guardar as listagens dos programas existentes na memória interna e no cartão de memória, para um ficheiro ASCII, que tem o prefixo “PRG” seguido da data e hora, e com extensão “txt”.

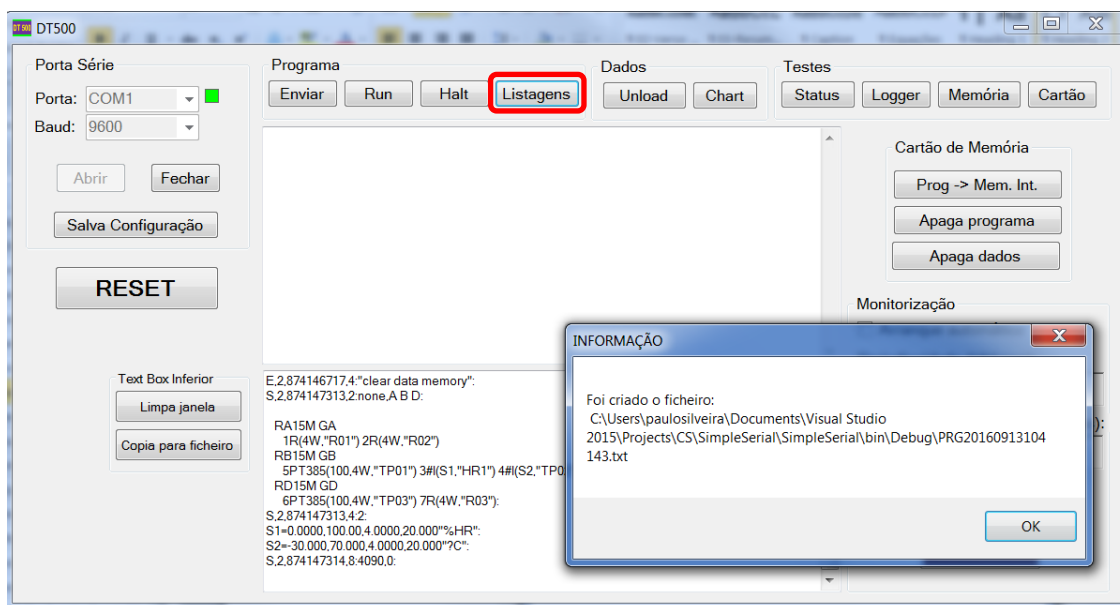


Figura 2.9 – Listagens dos programas

2.7 Gravação dos dados para ficheiro

No grupo de opções “Dados” existe o botão “Unload” (Figura 2.10) que desencadeia a gravação dos dados armazenados para um ficheiro ASCII do tipo “csv”, que contem a indicação da hora, da *schedule* e dos valores medidos, separados por ponto e vírgula.

O nome do ficheiro tem o prefixo “DT” seguido da data e hora. A extensão é, tal como se referiu “csv”.

É possível optar por descarregar os dados da memória interna, da memória do cartão, ou de ambas. É também possível escolher a totalidade, ou individualmente, as *schedules* a descarregar. Existe ainda a possibilidade de descarregar a totalidade dos dados, ou entre uma data inicial e uma data final.

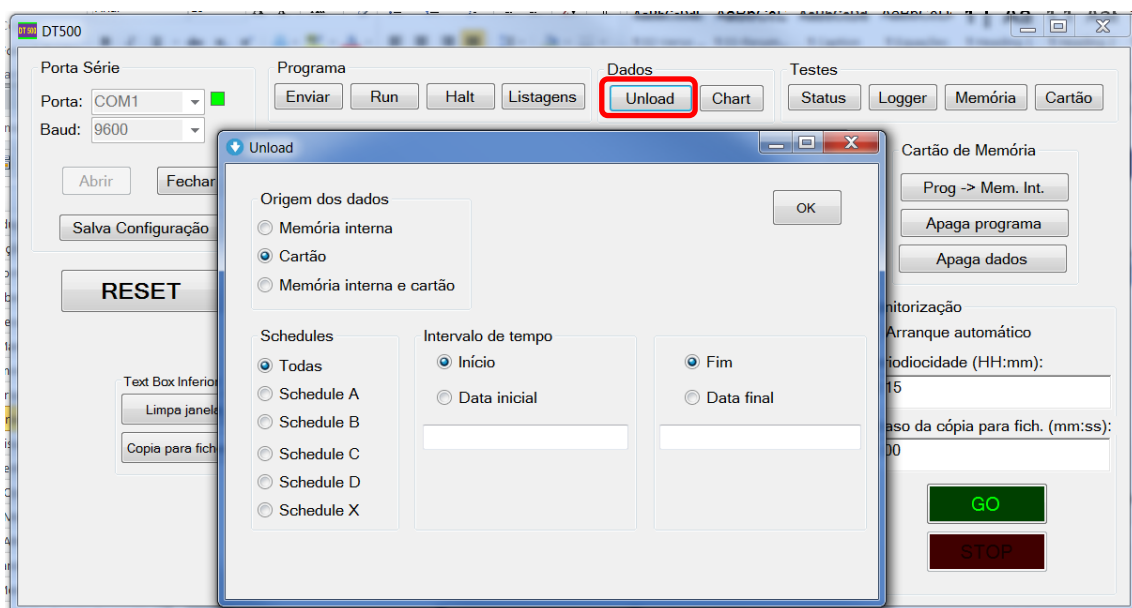


Figura 2.10 – Gravação dos dados armazenados para um ficheiro

2.8 Visualização das leituras em tempo real num gráfico

No grupo de opções “Dados” existe o botão “Chart” (Figura 2.11) que desencadeia a visualização das leituras num gráfico, em tempo real.

Para escolher os canais a visualizar selecionam-se os aparelhos nas *textboxes* das *schedules* e carrega-se no botão “Adicionar aparelho”. É também possível deixar de visualizar as leituras referentes a aparelhos previamente selecionados apagando-os da *textbox* que contem os aparelhos escolhidos e depois premindo o botão “Alterar”.

Para alterar os limites definidos automaticamente para o eixo das ordenadas, pressionam-se os botões direito ou esquerdo do rato, quando o cursor se encontrar na zona à esquerda deste eixo e preenche-se as caixas de texto, da *form* denominada “Limites do eixo das ordenadas”, com os limites pretendidos (Figura 2.12).

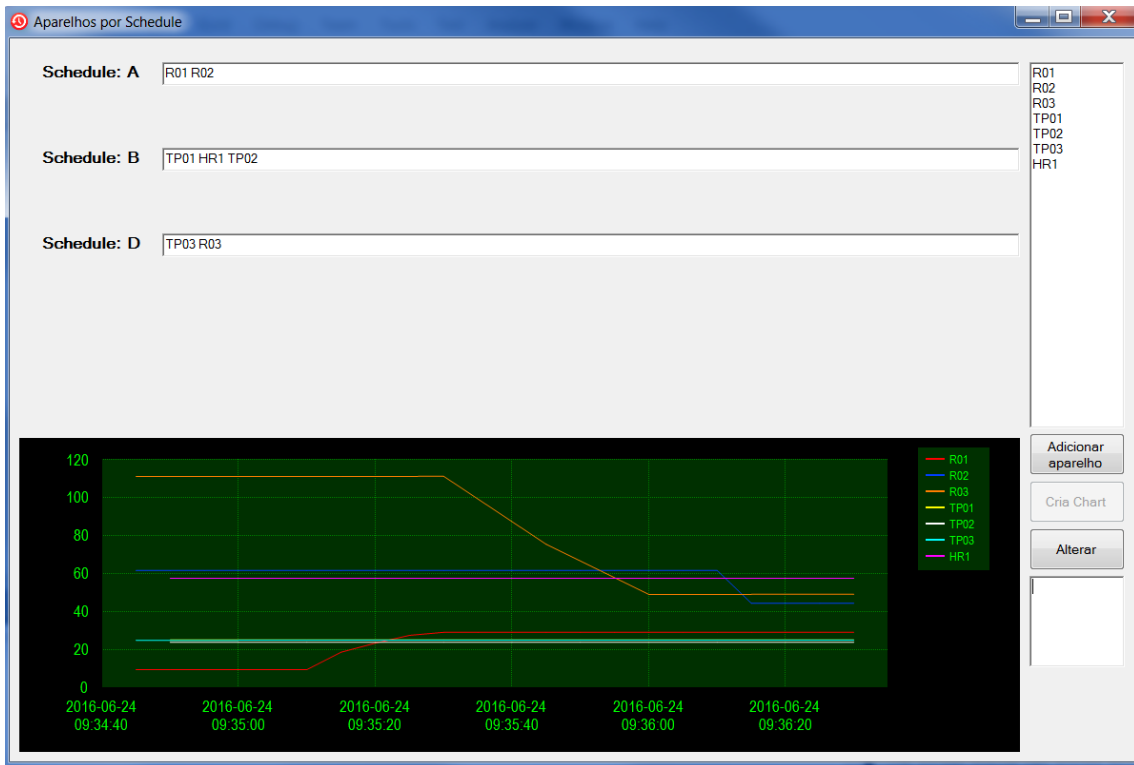


Figura 2.11 – Visualização dos dados em tempo real num gráfico

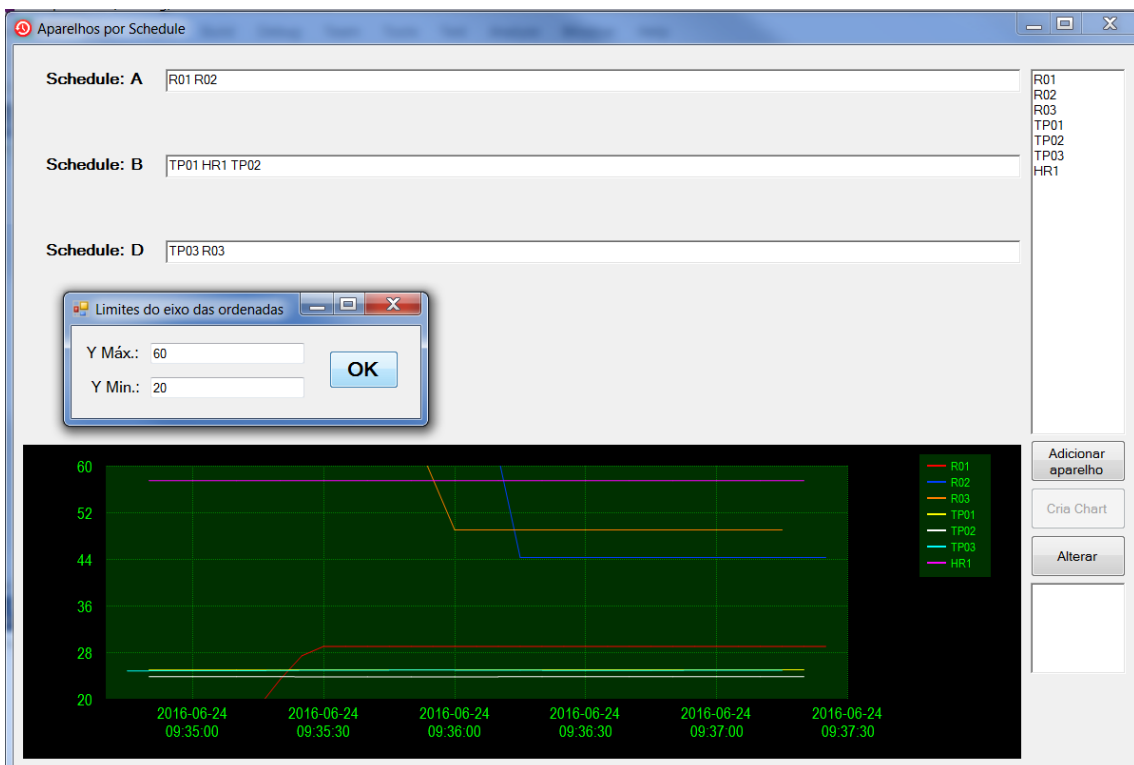


Figura 2.12 – Alteração da escala das ordenadas do gráfico

2.9 Testes ao *logger* e à informação armazenada

No grupo de opções “Testes”, existem os botões “Status”, “Logger”, “Memória” e “Cartão”.

O botão “Status” (Figura 2.13) possibilita conhecer:

- o endereço do *logger*;
- a versão da ROM;
- quais as *schedules* definidas e se estão ou não ativas;
- quantos alarmes se encontram ativos e inativos;
- a memória interna livre e ocupada;
- a memória do cartão livre e ocupada;
- a memória do programa livre e ocupada.

Se as *schedules* definidas se encontrarem ativas a luz que se encontra ao seu lado tem a cor verde, caso contrário tem a cor laranja.

O botão “Logger” (Figura 2.14) permite efetuar um teste ao *hardware* do *logger*. Se os parâmetros se encontrarem dentro dos limites definidos acende-se uma luz verde, de outro modo acende-se uma luz vermelha. É ainda apresentada a avaliação global ao *hardware* do *logger*.

O botão “Memória” (Figura 2.15) possibilita conhecer a quantidade de memória interna, ou do cartão, que se encontra livre ou ocupada.

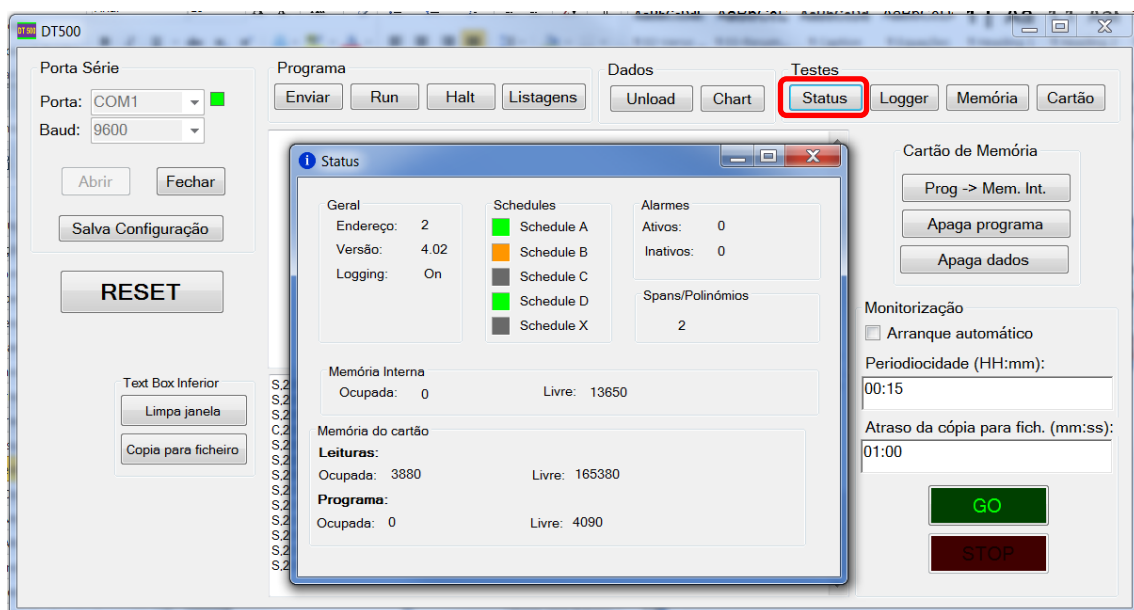


Figura 2.13 – Teste ao status do *logger*

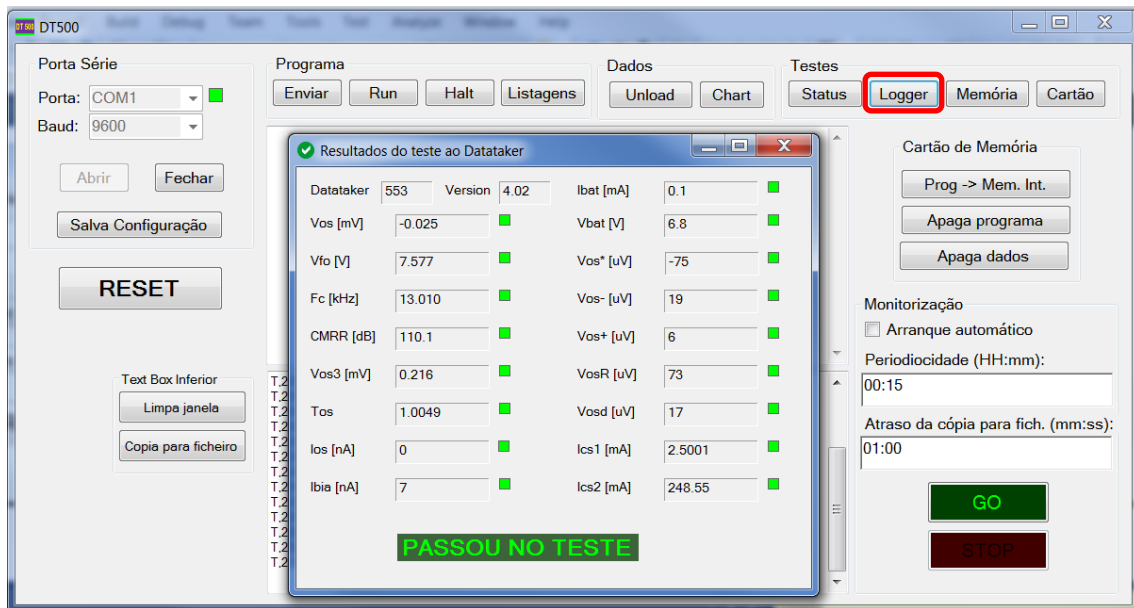


Figura 2.14 – Teste ao hardware do logger

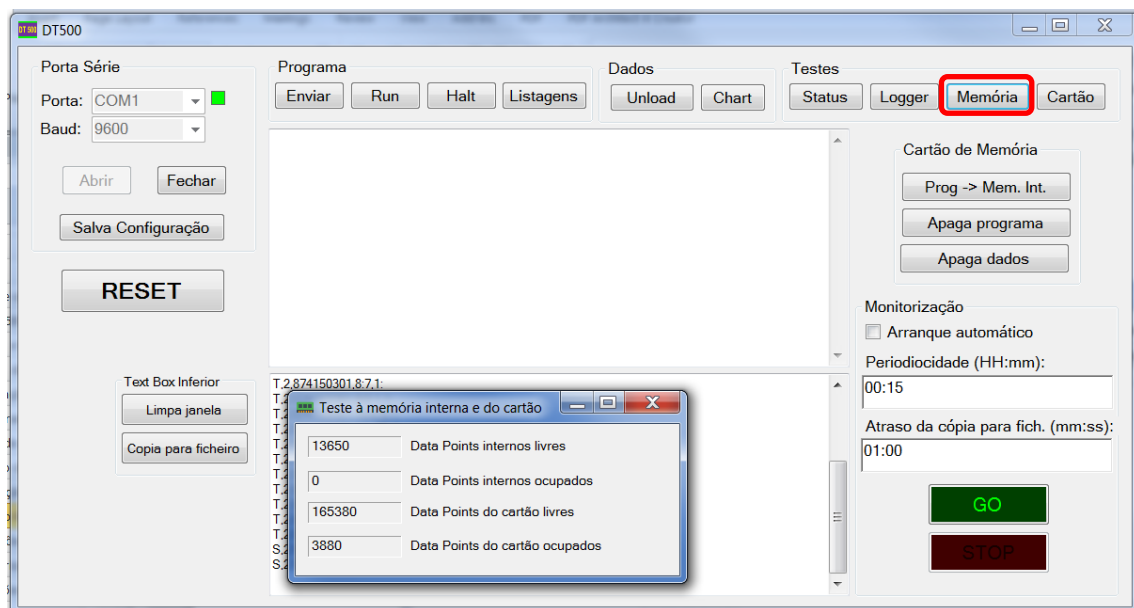


Figura 2.15 – Teste à memória do logger e do cartão

O botão de teste ao cartão desencadeia, não só o teste, mas também a formatação do cartão, pelo que, todos os dados e programa existentes no cartão de memória serão apagados.

À medida que o teste evolui a *TextBox* inferior (Figura 2.16) vai sendo preenchida por pontos finais, após o que, é escrito o resultado.

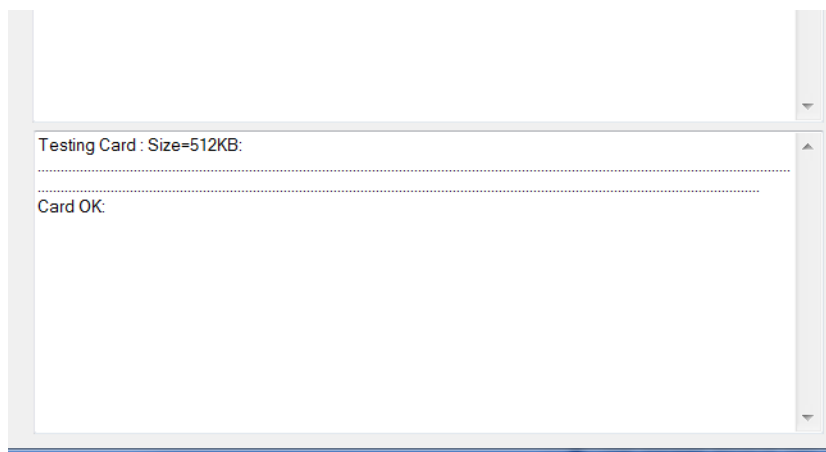


Figura 2.16 – Teste ao cartão de memória

2.10 Operações relativas ao cartão de memória

No grupo de opções “Cartão de memória”, existem os botões “Prog -> Mem. Int.”, “Apaga programa” e “Apaga dados” (Figura 2.17).

O botão “Prog -> Mem. Int.” faz com que o programa existente no cartão seja copiado para a memória interna. Se já existir um programa na memória interna o programa do cartão é acrescentado ao existente.

O botão “Apaga programa” faz com que seja apagado o programa existente no cartão.

O botão “Apaga dados” faz com que sejam apagados os dados existentes no cartão. Este botão é particularmente útil para limpar os cartões quando há a certeza que os dados já foram copiados para o computador.

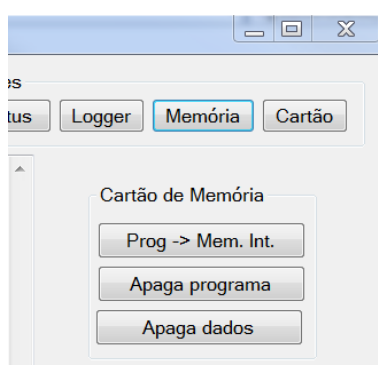


Figura 2.17 – Operações relativas ao cartão de memória

2.11 Monitorização

No grupo de opções “Monitorização” (Figura 2.18) existe uma *check box* que, quando se encontra marcada, permite o arranque automático da monitorização. Existe ainda a caixa de texto

“Periodicidade” onde se indica a periodicidade das leituras, sendo que essa periodicidade deverá ser igual ao menor valor atribuído às *schedules* e que foi definido no programa que se encontrar a correr. Existe também a caixa de texto “Atraso da cópia para fich.”, onde se indica o intervalo de tempo que deverá decorrer entre a leitura de valores e a sua escrita para ficheiro.

Os botões “GO” e “STOP” permitem, respetivamente, desencadear e parar o processo de registo das leituras efetuadas num ficheiro.

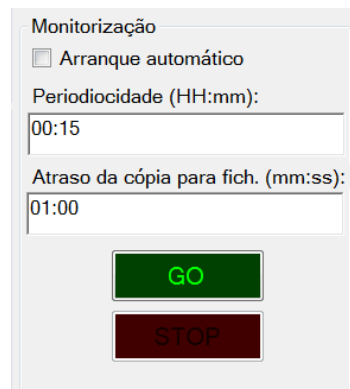


Figura 2.18 – Monitorização

O nome dos ficheiros criados para armazenar as leituras efetuadas no âmbito da monitorização de estruturas são constituídos por “DT” seguido da letra correspondente a uma das *schedules* ativas (A, B, C ou D) e completado com a data e hora no formato “@aaaaMMddHH”. A extensão é “txt” (ex.: DTA@2016092711.txt).

2.12 Alterações ao programa destinadas à utilização na monitorização de estruturas

Para efetuar a monitorização automática de estruturas é necessário que a aplicação faça um arranque automático, o que pode ser obtido colocando um atalho do executável na pasta “C:\Users\”[nome do utilizador](#)”\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup”.

Para além do arranque automático existe um ficheiro `pref.xml`, que pode ser alterado, por exemplo, com o NotePad e cujo conteúdo tipo é o seguinte:

```
<?xml version="1.0" encoding="UTF-8"?>
<Preferences xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <BaudRate>9600</BaudRate>
  <NomePorta>COM1</NomePorta>
  <Periodo>00:15</Periodo>
  <Delay>01:00</Delay>
  <ModoAuto>false</ModoAuto>
</Preferences>
```

Nestas linhas facilmente se identificam os parâmetros a alterar, designadamente o baud rate, o nome da porta, a periodicidade das leituras, o atraso com que são escritos os valores para um ficheiro e se se pretende que a aplicação faça o arranque automático.

Se o ficheiro pref.xml não existir, ele pode ser criado carregando no botão “Salva Configuração” do grupo de opções “Porta Série” (Figura 2.2).

Para efetuar o arranque automático é necessário marcar a *check box* “Arranque automático” do grupo de opções “Monitorização” (Figura 2.18).

Há no entanto alterações mais específicas que obrigam a alterações no código do programa. Nestas alterações incluem-se a localização do ficheiro de configuração pref.xml e o nome dos ficheiros para armazenar os dados obtidos na monitorização.

Para alterar a localização do ficheiro pref.xml é necessário editar o ficheiro app.config cuja listagem tipo é a seguinte:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="prefFile" value=".\\pref.xml" />
  </appSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
  </startup>
</configuration>
```

A localização encontra-se definida no texto marcado a amarelo.

Quando o ficheiro pref.xml não existe, os valores por omissão para o baud rate, porta, periodicidade das leituras, atraso na escrita para ficheiro, e arranque automático, estão definidos no ficheiro Preferences.cs, e poderão ser alterados na zona do código que se seguidamente transcreve e que se marca a amarelo:

```
[Serializable]
public class Preferences
{
    public Preferences()
    {
    }

    //Preferencias quando não existe o ficheiro de preferencias (pref.xml)
    private int _baudrate = 9600;
    private string _nomePorta = "COM1";
    private string _periodo = "00:15";
    private string _delay = "01:00";
    private bool _modoAuto = false;
```

3 / Comandos de programação do *logger*

A programação das leituras é efetuada enviando um ficheiro ASCII (*.txt) para a porta série do *data logger*.

Apresenta-se na Figura 3.1 um exemplo de programa destinado a ser enviado para a memória interna do *logger*, que se encontra diretamente ligado à porta série.

```
' Prog4.txt
H
CSCANS
CLEAR
P14=600 P17=120 P22=59 P24=13
P26=30 P31=1 P39=0 P36=0
/c/D/e/H/L/n/O/Q/R/S/T/u
S1=0,100,4,20"%HR"
S2=-30,70,4,20"C"
BEGIN
  RA5S HA
    1R(4W,"R01") 2R(4W,"R02")
  RB10S HB
    5PT385(100,4W,"TP01") 3#I(S1,"HR1") 4#I(S2,"TP02")
END
LOGON
```

Figura 3.1 – Programa para ser enviado para a memória interna

No caso de se pretender enviar o programa para o cartão de memória é apenas necessário iniciar as linhas com “;”. O programa deverá depois ser transferido para a memória interna utilizando os botões de “RESET” ou “Prog. -> Mem. Int.”.

```
;' Prog5.txt
;CSCANS
;CLEAR
;P14=600 P17=120 P22=59 P24=13
;P26=30 P31=1 P39=0 P36=0
;/c/D/e/H/L/n/O/Q/R/S/T/u
;
;S1=0,100,4,20"%HR"
;S2=-30,70,4,20"C"
;BEGIN
; RA5S GA
;   1R(4W,"R01") 2R(4W,"R02")
; RB10S GB
;   5PT385(100,4W,"TP01") 3#I(S1,"HR1") 4#I(S2,"TP02")
; RD15S GD
;   6PT385(100,4W,"TP03") 7R(4W,"R03")
;END
;LOGON
```

Figura 3.2 – Programa para ser enviado para o cartão de memória

Nos subcapítulos seguintes serão apresentados alguns exemplos de programação do *data logger* DT500, para diversos tipos de aparelhos mais vulgarmente utilizados.

3.1 Medição de temperaturas com termómetros do tipo PT100

Na Figura 3.3 apresentam-se alguns exemplos de medição de temperaturas com termómetros do tipo PT100. A quarta linha corresponde à utilização de uma ligação com quatro fios e as quinta e sexta linhas, respetivamente, à utilização de ligações com três e dois fios. A utilização destes dois últimos tipos de ligação tem instruções idênticas.

```
'Temperaturas PT385
BEGIN
  RA1S HA
    1PT385("TP01",100,4W) 'PT385 4W 0°C ->100 Ohm
    2PT385("TP02",100) 'PT385 2 ou 3 W 0°C ->100 Ohm
    3PT385("TP03",100)
END
```

Figura 3.3 – Medição de temperaturas com termómetros do tipo PT100

3.2 Medição de rotações com clinómetros Schaevitz

Na Figura 3.4 apresenta-se um exemplo de medição de rotações, nas duas direções, com um clinómetro bidirecional da Schaevitz.

```
'Clinómetro biaxial (Schaevitz) +-3 graus
S1=-10800,10800,-5,5"SegSex"
BEGIN
  RA1S HA
    1HV("CL01X",ES5,S1)
    2HV("CL01Y",ES5,S1)
END
```

Figura 3.4 – Medição de rotações com clinómetros Schaevitz

3.3 Medição de extensómetros de corda vibrante

Na Figura 3.5 apresenta-se um exemplo de medição de extensões com extensómetros de corda vibrante.

```
'Extensómetro de corda vibrante
Y11=0,0,0.0036"10-6"
BEGIN
  RA1S HA
    1FW("Ext.01",200,Y11)
END
```

Figura 3.5 – Medição de extensómetros de corda vibrante

3.4 Medição de deslocamentos com defletómetro potenciométrico (I)

Na Figura 3.6 apresenta-se um exemplo de medição de deslocamentos com um defletómetro potenciométrico que tem uma saída analógica de 4 a 20 mA.

```
'Defletómetro potenciométrico (JX-P420)
S2=0,500,4,20"mm"
BEGIN
  RA1S HA
  1#I(S2,"DV1")
END
```

Figura 3.6 – Medição de deslocamentos com defletómetro potenciométrico (I)

3.5 Medição de deslocamentos com defletómetro magnetostritivo (I)

Na Figura 3.7 apresenta-se um exemplo de medição de deslocamentos com um defletómetro magnetostritivo que tem uma saída analógica de 4 a 20 mA.

```
'Magnetostritivos (corrente)
S2=0,500,4,20"mm"
BEGIN
  RA1S HA
  1#I("DH01",S2)
END
```

Figura 3.7 – Medição de deslocamentos com defletómetro magnetostritivo (I)

3.6 Medição de deslocamentos com defletómetro magnetostritivo (V)

Na Figura 3.8 apresenta-se um exemplo de medição de deslocamentos com um defletómetro magnetostritivo que tem uma saída analógica de 0 a 10 V.

```
'Magnetostritivos (Tensão)
S2=0,500,0,10"mm"
BEGIN
  RA1S HA
  1*HV ("DH01",S2)
END
```

Figura 3.8 – Medição de deslocamentos com defletómetro magnetostritivo (V)

3.7 Medição de pressão

Na Figura 3.9 apresentam-se alguns exemplos para medição de pressões com células de pressão.

A célula da Schedule A é medida em tensão (0 a 5 V) e a da Schedule B em corrente (4 a 20 mA).

```
'Células de Pressão (RA: Provetes de fluência – S3; RB: níveis líquidos – S4)
S3=-1.86,98.14,0,5"bar"
S4=0,3500,4,20"mm"

BEGIN
  RA1S HA
    1*HV("PF01",S3)
  RB1S HB
    2#I("NL8",S4)
END
```

Figura 3.9 – Medição de pressão

3.8 Medição de extensões com extensômetros em ponte completa

Na Figura 3.10 apresenta-se um exemplo de medição de extensões com uma ponte completa de extensômetros de 350 Ω .

```
'Extensômetros elétricos de resistência em ponte completa
S5=0,746.826,0,1000"us"
BEGIN
  RA1S HA
    1BGI("Ext01",350,4W,S5)
END
```

Figura 3.10 – Medição de extensões com extensômetros elétricos de resistência em ponte completa

4 | Descrição do software DT500Com

Na Figura 4.1 exibe-se uma imagem do *Solution Explorer* do software DT500Com. Este programa é constituído por diversos ficheiros que contêm a definição do código e dos métodos utilizados.

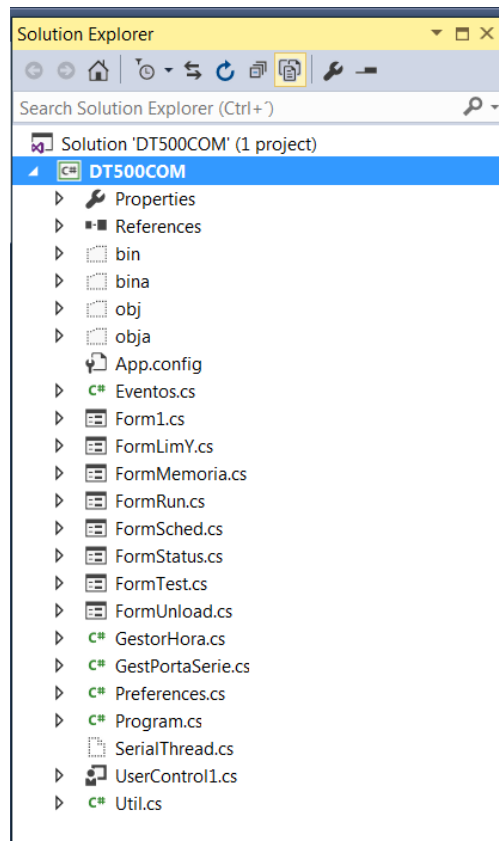


Figura 4.1 – Imagem do *Solution Explorer*

No ponto 4.1 deste capítulo faz-se a introdução a alguns conceitos da linguagem de programação C# e no ponto 4.1 descreve-se o código desenvolvido.

4.1 Introdução a alguns conceitos da linguagem C#

A linguagem de programação C foi desenvolvida entre 1969 e 1973 por Dennis Ritchie e resulta da evolução da linguagem B, ambas criadas na Bell Labs.

Em 1983, com base na linguagem C, Bjarne Stroustrup criou o C++. Este nome resultou da associação da letra C ao operador incremental ++, significando portanto uma evolução na linguagem C.

Mais recentemente, em julho de 2000, e em consequência do desenvolvimento, pela Microsoft, do ambiente .NET, foi criada a linguagem de programação C#, cuja autoria é de Anders Hejlsberg e

Scott Wiltamuth. Esta linguagem, que apresenta semelhanças com a linguagem Java, é no entanto mais parecida com o C++.

O nome C# (C Sharp) resulta de uma analogia com a notação musical. O símbolo sharp (#) designado em português por sustenido, destina-se a elevar a altura da nota que se lhe segue, em um semitom, e como tem igualmente a particularidade de se assemelhar à associação de quatro sinais "+", permite reforçar a ideia de que a linguagem C# resulta da evolução da linguagem C++.

O C#, tal como outras linguagens que permitem a programação orientada por objetos, assenta em quatro pilares que são, a abstração, o encapsulamento, a herança e o polimorfismo.

Entre o ponto 4.1.1 e o ponto 4.1.4 apresentam-se alguns conceitos básicos relacionados com estas características.

4.1.1 Abstração

Nas linguagens orientadas por objetos existem dois conceitos fundamentais: classes e objetos, sendo as classes um modelo para a definição de objetos, isto é, uma classe permite criar uma abstração de um conceito do mundo real. A criação de um objeto a partir de uma classe corresponde à criação de uma instância dessa classe, sendo que as características que representam um objeto se designam como propriedades.

Para além dos objetos, nas classes podem também definir-se métodos, que correspondem a ações que os objetos podem executar. O conjunto composto pelos métodos e propriedades que constituem uma classe designa-se como os membros da classe.

Os membros de uma classe podem ser do tipo public, private ou protected, em função do nível de acesso que têm para as outras entidades, conferindo as seguintes características:

- public: o método ou propriedade podem ser acedidos dentro do próprio objeto ou a partir de qualquer objeto externo;
- private: o método ou propriedade só podem ser acedidos dentro do mesmo objeto;
- protected: método ou propriedade só podem ser acedidos dentro do mesmo objeto, ou de um objeto que derive daquela classe;

Apesar das classes serem modelos para a criação de objetos, é possível aceder a uma classe, ou utilizar um método de uma classe, sem criar um objeto. Para tal utiliza-se o modificador static, criando classes e métodos estáticos. Este procedimento só se utiliza quando não é necessário criar mais do que um objeto da mesma classe.

Desta forma, seguindo os princípios de abstração, é possível "esconder" a complexidade interna de cada objeto, isolando essa complexidade através dos níveis de acesso para o exterior. Toda a complexidade das classes fica isolada na sua definição e caracterização.

4.1.2 Encapsulamento

O encapsulamento corresponde à técnica que permite que os detalhes internos do funcionamento dos métodos de cada classe sejam específicos da definição da classe e não de cada um dos objetos. Desta forma, o conhecimento sobre o detalhe de implementação de cada método é desconhecido para os objetos, reduzindo a complexidade da programação que se concentra apenas na definição da classe.

Como resultado da técnica de encapsulamento, torna-se possível reduzir as alterações no código a um local, isto é, na definição dos membros e atributos da classe.

4.1.3 Herança

O conceito de herança equivale à sua definição no mundo real (numa relação pai-filho), isto é, uma *classe-filha* pode herdar características (propriedades e/ou métodos) de uma *classe-pai*. No caso das linguagens de programação, apenas os métodos e propriedades que sejam *public* ou *protected* são herdados pelas *classe-filha*.

A herança permite efetuar uma reutilização do código desenvolvido para outras classes, numa classe nova, especializando e estendendo apenas o conjunto de propriedades e métodos que são específicos da nova classe (classe-filha).

A herança permite que classes mais abstratas representem o comportamento de classes mais concretas. Por exemplo, num cenário em que existem vários tipos de *data logger*, pode haver uma classe abstrata (classe pai), que representa o comportamento de todas as classes específicas (cada um dos tipos de *data logger*).

4.1.4 Polimorfismo

O polimorfismo consiste na possibilidade de alterar o funcionamento de métodos. Um caso corrente de utilização do polimorfismo é a definição de vários métodos com o mesmo nome. Nestes casos a escolha da versão a utilizar será definida pelas variáveis de entrada ou de saída. Um exemplo corresponde à definição de métodos denominados “Soma(a, b)”, que fazem uma soma se a e b forem duas variáveis numéricas, ou uma concatenação se a e b forem strings.

O polimorfismo pode ser estático, se a resposta de uma função é definida quando da compilação, ou dinâmico quando a resposta é definida em *runtime*.

4.2 Código

Tal como se referiu, o *software* desenvolvido contém o código repartido por vários ficheiros. Nos subcapítulos seguintes resumem-se os aspetos mais importantes relativos a esses ficheiros.

4.2.1 Ficheiro Form1.cs

O ficheiro Form1.cs contém o código com que o programa inicia e um conjunto de métodos cuja função se resume no Quadro 4.1.

Quadro 4.1 – Métodos definidos em Form1.cs

Método	Invocado	Finalidade
Form1_Load	Sim	Carrega a Form1 e permite o arranque automático do programa
Form1_FormClosing	Sim	Fecha a Form1, a thread "thrdPortaSerie" e os streams dos ficheiros abertos
DataRecebida	Sim	É um handler do evento correspondente a receber dados na porta série
DataReceivedSync	Sim	Guarda os dados recebidos na porta série e copia-os para a texto box inferior
buttonStart_Click	Botão Abrir	Abre a porta série escolhida
buttonStop_Click	Botão Fechar	Fecha a porta série e os ficheiros abertos
textBox1_KeyPress	Evento	Escreve os caracteres da texto box superior para a porta série
buttonSendProg_Click	Botão Enviar	Envia o programa para o <i>logger</i>
buttonRun_Click	Botão Run	Faz correr o programa existente no <i>logger</i>
buttonHalt_Click	Botão Halt	Faz parar o programa que está a correr no <i>logger</i>
buttonUnload_Click	Botão Unload	Faz o unload dos dados existentes no <i>logger</i> para um ficheiro
buttonTest_Click	Botão Logger	Faz o teste ao <i>hardware</i> do <i>logger</i>
btnMemoria_Click	Botão Memória	Mostra a memória utilizada
buttonStatus_Click	Botão Status	Mostra o endereço do <i>logger</i> , a versão da ROM, quais as schedules definidas e se estão ou não ativas, quantos alarmes se encontram ativos e inativos, a memória interna livre e ocupada, a memória do cartão livre e ocupada, a memória do programa livre e ocupada.
buttonCanais_Click	Botão Chart	Permite fazer gráficos dos valores medidos em tempo real
btnListagem_Click	Botão Listagem	Escreve a listagem do programa existente em memória para a texto box inferior e para um ficheiro
buttonCartao_Click	Botão Cartão	Faz o teste e a formatação do cartão de memória
btnRPprog_Click	Botão Prog -> Mem. Int	Copia o programa do cartão de memória para a memória interna do <i>logger</i>

Método	Invocado	Finalidade
btnCProg_Click	Botão Apaga programa	Apaga o programa do cartão de memória
btnCData_Click	Botão Apaga dados	Apaga os dados do cartão de memória
btnReset_Click	Botão RESET	Faz o reset ao <i>logger</i> , apaga os dados e o programa existentes
buttonLimpa_Click	Botão Limpa janela	Limpa a text box inferior
buttonCopia_Click	Botão Copia para ficheiro	Copia o conteúdo da text box inferior para um ficheiro
btnGo_Click	Botão GO	Arranca com a monitorização periódica
btnStop_Click	Botão STOP	Para a monitorização periódica
JaTaNaHora	Evento	É um handler do evento correspondente a se estar na hora de fazer leituras
TaNaHoraSync	Sim	Faz o unload das últimas leituras para ficheiros quando acontece o evento JaTaNaHora
CloseFichMonit	Sim	Fecha os ficheiros da monitorização
DisplayText	Sim	Escreve na text box inferior os dados que chegam à porta série, dando-lhes tratamento diferente consoante o comando que solicitou a resposta.
setDefaultComboValues	Sim	Preenche a comboBoxBaud e define o seu valor por default (9600) e também a portNameComboBox e define o seu valor por default ("COM1")

4.2.2 Ficheiro FormLimY.cs

O ficheiro FormLimY.cs contém um conjunto de métodos que permite alterar a escala das ordenadas dos gráficos desenhados em tempo real, resumindo-se a função no Quadro 4.2.

A alteração das escalas é feita na form FormLimY.

Quadro 4.2 – Métodos definidos em FormLimY.cs

Método	Invocado	Finalidade
FormLimY_Load	Sim	Carrega a FormLimY para alterar a escala das ordenadas. Atribui um valor ao boleano btnclick que é utilizado em FormSched.cs
FormLimY_FormClosing	Sim	Fecha a FormLimY. Atribui um valor ao boleano btnclick que é utilizado em FormSched.cs
btnLimYOk_Click	Botão OK	Altera os limites do eixo das ordenadas. Atribui um valor ao boleano btnclick que é utilizado em FormSched.cs

4.2.3 Ficheiro FormMemoria.cs

O ficheiro FormMemoria.cs contém um método que permite dar a conhecer a memória livre e ocupada, resume-se a função dos métodos existentes no Quadro 4.3.

Quadro 4.3 – Métodos definidos em FormMemoria.cs

Método	Invocado	Finalidade
FormMemoria_Load	Sim	Carrega a FormMemoria
preencheFormMemoria	Form1.cs	Preenche a FormMemoria no método DisplayText da Form1.cs

4.2.4 Ficheiro FormRun.cs

O ficheiro FormRun.cs contém um conjunto de métodos que permite fazer correr o programa existente na memória interna do *logger*. Resumem-se as funções dos vários métodos no Quadro 4.4.

Na form FormRun existe um botão OK para desencadear a execução dos horários escolhidos do programa.

Quadro 4.4 – Métodos definidos em FormRun.cs

Método	Invocado	Finalidade
rdBtnTotSched_CheckedChanged	Radio button	Escolhe todas as schedules
rdnBtnEscSched_CheckedChanged	Radio button	Permite escolher individualmente as schedules
buttonOK_Click	Botão OK	Faz executar as schedules escolhidas
chkBoxA_CheckedChanged	Evento	Escolhe a Schedule A
chkBoxB_CheckedChanged	Evento	Escolhe a Schedule B
chkBoxC_CheckedChanged	Evento	Escolhe a Schedule C
chkBoxD_CheckedChanged	Evento	Escolhe a Schedule D
InicializaForm	Sim	Faz a inicialização da form FormRun

4.2.5 Ficheiro FormSched.cs

O ficheiro FormSched.cs contém um conjunto de métodos que permite construir um gráfico para visualizar os valores medidos em tempo real. Resumem-se as funções dos vários métodos no Quadro 4.5.

Na form FormSched existe um botão para criar o gráfico, um botão para adicionar os aparelhos que se pretende visualizar e um botão para alterar os aparelhos escolhidos. Existem duas text boxes, uma com os aparelhos que podemos visualizar e outra com os aparelhos que escolhemos.

Quadro 4.5 – Métodos definidos em FormSched.cs

Método	Invocado	Finalidade
FormSched_Load	Sim	Carrega a form onde se vai desenhar a chart
btnEscolha_Click	Botão Adicionar aparelho	Permite escolher os aparelhos cujos valores se pretende representar
btnCriaChart_Click	Botão Cria Chart	Cria a chart
FormSched_FormClosing	Evento	Fecha a form FormSched
btnAltera_Click	Botão Alterar	Valida as alterações introduzidas na lista dos aparelhos escolhidos
ButtonDown	Evento	Desencadeia um evento quando se prime um dos botões do rato
_chart_MouseMove	Evento	Mostra as coordenadas X,Y do gráfico quando o cursor do rato se aproxima da curva
_chart_GetToolTipText	Evento	Permite obter as coordenadas X,Y do gráfico
preencheFormSched	Form1.cs	Cria e preenche as text boxes com os aparelhos das schedules existentes na FormSched no método DisplayText da Form1.cs
dicCanEscolhidos	Form1.cs	Determina o n.º de aparelhos escolhidos por schedule e preenche: o Dictionary dicTotCan é constituído pela letra que identifica a schedule e por uma lista de strings com os nomes dos aparelhos lidos em cada horário; o Dictionary dicCanEsc é constituído pela letra que identifica a schedule e por uma lista de integers com o índice do aparelho escolhido na respetiva Schedule; a List de strings chnEsc com os nomes dos aparelhos escolhidos para a chart

4.2.6 Ficheiro FormStatus.cs

O ficheiro FormStatus.cs contém um método que permite conhecer os horários definidos e saber o seu estado, assim como a situação da memória existente no *logger* e no cartão de memória. Resumem-se as funções dos métodos existentes no Quadro 4.6.

Quadro 4.6 – Métodos definidos em FormStatus.cs

Método	Invocado	Finalidade
FormStatus_Load	Sim	Carrega a form FormStatus
preencheFormStatus	Form1.cs	Preenche a informação disponibilizada na FormStatus no método DisplayText da Form1.cs

4.2.7 Ficheiro FormTest.cs

O ficheiro FormTest.cs contém um método que permite fazer um teste ao *hardware* do *logger*. Resumem-se as funções dos métodos existentes no Quadro 4.7.

Quadro 4.7 – Métodos definidos em FormTest.cs

Método	Invocado	Finalidade
FormTest_Load	Sim	Carrega a form FormTest
preencheFormTest	Form1.cs	Preenche a informação resultante do teste ao <i>hardware</i> na FormTest no método DisplayText da Form1.cs

4.2.8 Ficheiro GestorHora.cs

O ficheiro GestorHora.cs contém diversos métodos a fazer a monitorização do comportamento estrutural, com leituras periódicas e o seu armazenamento em ficheiros. Resumem-se as funções dos métodos existentes no Quadro 4.8.

Quadro 4.8 – Métodos definidos em GestorHora.cs

Método	Invocado	Finalidade
AtualizaSched	Form1.cs	Faz a monitorização periódica das estruturas. Este método é chamado na FormTest no método btnGo_Click da Form1.cs
Start	Sim	Provoca o arranque da thread thrdHora
Stop	Sim	Provoca a paragem da thread thrdHora
AbreFicheiros	Sim Form1.cs	Abre os ficheiros com os valores da monitorização periódica das estruturas. Este método é chamado no método RunMethod de GestorHora.cs e no método DisplayText da Form1.cs
RunMethod	Sim	É chamado no método Start e controla o tempo de sleep da thread thrdHora, em função da periodicidade com que são feitas as leituras, para evitar o funcionamento excessivo do processador.

4.2.9 Ficheiro GestorPortaSerie.cs

O ficheiro GestorPortaSerie.cs contém diversos métodos para fazer a leitura e a escrita para a porta série, criando para o efeito a thread thrdPortaSerie. Resumem-se as funções dos métodos existentes no Quadro 4.9.

Quadro 4.9 – Métodos definidos em GestorPortaSerie.cs

Método	Invocado	Finalidade
OpenSerialPort	Form1.cs	Abre a porta série. Este método é chamado no método buttonStart_Click de Form1.cs

Método	Invocado	Finalidade
CloseSerialPort	Form1.cs	Fecha a porta série. Este método é chamado no método buttonStop_Click de Form1.cs
SerialPortIsOpen	!	Retorna um boleano que tem o valor true se a porta estiver aberta
InitLogger	Form1.cs	Abre a porta série. Este método é chamado no método buttonStart_Click de Form1.cs
StartReading	Form1.cs	Inicia a thread thrdPortaSerie. Este método é chamado no método buttonStart_Click de Form1.cs
StopReading	Form1.cs GestPortaSerie.cs	Para a thread thrdPortaSerie. Este método é chamado nos métodos Form1_FormClosing e buttonStop_Click de Form1.cs e também no método CloseSerialPort de GestPortaSerie.cs
EscrevePorta [Char]	Form1.cs	Escreve um caracter para a porta série. Este método é chamado no método textBox1_KeyPress de Form1.cs
EscrevePorta [string - overload]	Form1.cs FormRun.cs FormUnload.cs	Escreve um string para a porta série. Este método é chamado em Form1.cs nos métodos: buttonSendProg_Click, buttonTest_Click, btnMemoria_Click, buttonStatus_Click, buttonCanais_Click, btnListagem_Click, buttonCartao_Click, btnRPprog_Click, btnCProg_Click, btnCData_Click, btnReset_Click, btnGo_Click, TaNaHoraSync. Em FormRun.cs no método buttonOK_Click. Em FormUnload.cs nos métodos BtnOK_Click e DatasLimite.
LePorta	FormUnload.cs	Destina-se a determinar as datas limite dos dados existentes no cartão ou no <i>logger</i> . Este método é chamado no método DatasLimite.
EnviaProg	Form1.cs	Faz o envio do programa para o <i>logger</i> ou para o cartão. Este método é chamado em Form1.cs no método buttonSendProg_Click.
RunMethod	Sim	Controla se há atividade na porta verificando se a linha anterior recebida foi modificada

4.2.10 Ficheiro Preferencias.cs

O ficheiro Preferencias.cs contém um método que permite fazer um teste ao *hardware* do *logger*. Resumem-se as funções dos métodos existentes no Quadro 4.1.

Quadro 4.10 – Métodos definidos em Preferencias.cs

Método	Invocado	Finalidade
Save	Form1.cs	Salva a configuração no método btnSalvaConf_Click da Form1.cs
Load	Form1.cs	Carrega a configuração no método setDefaultComboValues da Form1.cs

4.2.11 Ficheiro SerialThread.cs

O ficheiro SerialThread.cs contém diversos métodos relacionados com a leitura e a escrita para a porta série, criando para o efeito a thread thrdPortaSerie. Resumem-se as funções dos métodos existentes no Quadro 4.11.

Quadro 4.11 – Métodos definidos em SerialThread.cs

Método	Invocado	Finalidade
Close	!	Fecha a porta série e limpa os buffers.
EscrevePorta	!	Escreve um caracter para a porta série
RunMethod	Sim	Abre a porta série e controla se há atividade verificando se a linha anterior recebida foi modificada

4.2.12 Ficheiro Util.cs

O ficheiro Util.cs contém diversos métodos relacionados com a leitura e a escrita para a porta série, e que se destinam . Resumem-se as funções dos métodos existentes no Quadro 4.12.

Quadro 4.12 – Métodos definidos em Util.cs

Método	Invocado	Finalidade
separaStrCanais	Form1.cs	Faz a separação (parse) do string de resposta ao comando "STATUS10", para determinar os nomes das grandezas associadas às leituras efetuadas. A coleção dicCanais é do tipo "Dictionary" e é composta por um string com a letra que identifica a schedule e uma "List" com os nomes dos aparelhos lidos nessa Schedule. Este método é chamado no método DisplayText de Form1.cs.
GetDouble	Sim FormLimY.cs	Converte string em double independentemente do separador dos decimais ser "." ou ",". Este método é chamado no método AddXYChart e no método btnLimYOk_Click de FormLimY.cs.
AddXYChart	Form1.cs	Adiciona pontos à chart à medida que os valores vão sendo enviados pelo <i>logger</i> e vai fazendo o ajuste automático dos limites dos eixos. Este método é chamado no método DisplayText de Form1.cs.
canaisTodos	!	Cria uma string list com a totalidade dos canais existentes.
AlteraEscY	FormLimY.cs	Altera os valores máximo e mínimo do eixo das ordenadas da chart. Este método é chamado no método btnLimYOk_Click de FormLimY.cs.
ConfiguraChart	FormSched.cs	Configura a chart. Este método é chamado no método btnCriaChart_Click de FormSched.cs.

Método	Invocado	Finalidade
canaisChart	FormSched.cs	Faz a separação (parse) do string existente na text box da form FormSched que contem os canais escolhidos para incluir na chart e coloca os nomes dos canais escolhidos numa List de strings. Este método é chamado nos métodos btnCriaChart_Click e btnAltera_Click de FormSched.cs.
separaString	FormMemoria.cs FormStatus.cs FormTest.cs	Separa os strings, pela mudança de linha, num array de strings. Apaga os "LF", localiza a primeira ocorrência de "." para conservar apenas os dois últimos valores. Este método é chamado nos métodos preencheFormMemoria de FormMemoria.cs, preencheFormStatus de FormStatus.cs e preencheFormTest de FormTest.
ConvStringSec	Sim	Converte a data enviada pelo logger, que se encontra no formato diferença de segundos para 1989, para uma data e hora em formato datetime. Este método é chamado no método DataHora de Util.cs
DataHora	FormUnload.cs	Trata os strings que resultam do comando "UNLOAD". Apaga os "CR" e os "LF", localiza a primeira ocorrência de "." para conservar apenas os dois últimos valores. Este método é chamado no método DatasLimite de FormUnload.cs.
ExtraiValores	Form1.cs	Trata os strings que contêm as leituras que são escritos na texto box inferior da form Form1. Este método é chamado no método DisplayText de Form1.cs.
TrataString	Form1.cs	Trata os strings que vão sendo enviados pelo logger após efetuadas as leituras e também aqueles que resultam do comando "UNLOAD". Este método é chamado no método DisplayText de Form1.cs.
NomeFich	Form1.cs FormUnload.cs GestorHora.cs	Definição do nome do ficheiro onde são guardados os dados resultantes da escrita do programa existente em memória, da cópia do conteúdo da texto box inferior da form Form1, do unload dos dados e dos dados da monitorização estrutural. Os nomes começam respetivamente por "PRG", "TxBox", "DT" ou "[DTA, DTB, DTC e DTD]" e segue-se a data e a hora a que é criado. Os ficheiros são criados na mesma diretorio do programa. Este método é chamado no método btnListagem_Click e buttonCopia de Form1.cs, no método BtnOK_Click de FormUnload.cs e no método AbreFicheiros de GestorHora.cs.
MudaData	Form1.cs	Muda a data do formato "dd-MM-aaaa" para "aaaa-MM-dd". É utilizado unicamente quando está a escrever os ficheiros que resultam da monitorização estrutural periódica.

5 | Conclusões

A evolução natural do *software* DT500Com apresentado neste relatório será a incorporação da possibilidade de controlar um conjunto de *loggers* do tipo DT500 ligados em rede (RS485) e também a sua adaptação ao controlo de outro tipo de *loggers*, beneficiando da programação orientada por objetos que a linguagem C# permite. Desta forma, a inclusão de novos tipos de *loggers* necessita apenas que sejam definidas as particularidades de cada um dos *loggers*, através da criação da classe *logger* que encapsula o comportamento genérico de todos os *loggers* e da posterior extensão de cada um dos tipos de *logger* com a redefinição do comportamento e estrutura específicos.

Lisboa, LNEC, outubro de 2016

VISTOS

O Chefe do Núcleo de Observação de Estruturas



Manuel Pipa

AUTORIA



Paulo Silveira
Investigador Principal



José Barateiro
Investigador Auxiliar


O Diretor do Departamento de Estruturas



José Manuel Catarino

Chefe do Núcleo de Tecnologias da
Informação em Engenharia Civil

O Conselho Diretivo



Maria Alzira Santos
Vogal do Conselho Diretivo

Anexos

ANEXO I
Form1.cs


```

//DT500COM - Form1.cs
using System;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;
using System.Threading;
using System.Collections.Generic;
using System.Windows.Forms.DataVisualization.Charting;
using System.Globalization;
using System.Text.RegularExpressions;

namespace DT500COM
{
    public partial class Form1 : Form
    {
        private string RxString, ficheiro;
        private string straux = "";
        private string strTest = "";
        private string strMonit = "";
        private bool abrFichMonit = false;
        private bool btnTest = false;
        private bool btnMemo = false;
        private bool btnULoad = false;
        private bool btnStatus = false;
        private bool btnCns = false;
        private bool btnChart = false;
        private bool btnLista = false;
        private bool btnMonit = false;
        private bool segundaVez = false;
        private GestorPortaSerie portaCom = null;
        private StreamWriter file = null;
        private FormTest resultaTest = null;
        private FormMemoria resultaMemoria = null;
        private FormUnload unload = null;
        private FormRun runProg = null;
        private FormStatus status = null;
        private FormSched _horarios = null;
        private List<int> nChEsc = new List<int>();
        private Dictionary<string, List<int>> dicCanEsc = new Dictionary<string, List<int>>();
        private Dictionary<string, int> serieID = new Dictionary<string, int>();
        private Chart chart;
        private static Form1 _form1;
        private Dictionary<string, List<string>> _dicCanais;
        private GestorHora gHora = null;
        private List<StreamWriter> fichMonit = new List<StreamWriter>();
        private List<string> fichName = new List<string>();
        private static int[] contLin = { 0, 0, 0, 0 };
        private static string[] _strcanais = { "", "", "", "" };
        private DateTime horaFich;
        private DateTime horaAtual;
        private Preferences pref = new Preferences();

        public Form1()
        {
            InitializeComponent();
            setDefaultComboValues(); //Preenche os valores de default
        }

        #region Properties

        public static string[] strcanais
        {
            get { return _strcanais; }
            set { _strcanais = value; }
        }

        public FormSched horarios
        {
            get { return _horarios; }
            set { _horarios = value; }
        }

        public Dictionary<string, List<string>> dicCanais
        {

```

```

        get { return _dicCanais; }
        set { _dicCanais = value; }
    }

    public static Form1 form1
    {
        get { return _form1; }
        set { _form1 = value; }
    }

#endregion

#region Events

private void Form1_Load(object sender, EventArgs e)
{
    _form1 = this;

    //As linhas abaixo permitem o arranque automatico para a aquisição automática
    rchTextBoxPeri.Text = pref.Periodo; //periodicidade das leituras em hh:mm
    rchTextBoxAtras.Text = pref.Delay; //atraso da escrita das leituras para ficheiro em mm:ss
    cbAuto.Checked = pref.ModosAuto; //Check box marcada para fazer a monitorização automática

    if (pref.ModosAuto)
    {
        buttonStart.PerformClick();
        btnGo.PerformClick();
    }
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    //Destroi a "thrdPortaSerie", se já tiver sido criada no GestPortaSerie, quando fecha a
    Form1 no "X"
    if (portaCom != null) portaCom.StopReading();

    CloseFichMonit(); //Fecha os ficheiros da monitorização
    if (file != null) file.Close(); //Fecha o ficheiro do Unload se este estiver aberto
}

void DataRecebida(object s, EventArgs e)
{
    // Note: this method is called in the thread context, thus we must
    // use Invoke to talk to UI controls. So invoke a method on our
    // thread.
    Invoke(new EventHandler<DataEventArgs>(DataReceivedSync), new object[] { s, e });
}

void DataReceivedSync(object s, EventArgs e)
{
    RxString = ((DataEventArgs)e).Data;
    DisplayText();
}

private void buttonStart_Click(object sender, EventArgs e)
{
    if (portaCom == null)
    {
        portaCom = new GestorPortaSerie();
        portaCom.DataReceived += DataRecebida;
    }

    //portaCom.OpenSerialPort(portNameComboBox.SelectedItem.ToString(),
    Convert.ToInt32(comboBoxBaud.SelectedItem));

    if (pref.ModosAuto)
    {
        portaCom.OpenSerialPort(pref.NomePorta, pref.BaudRate);
    }
}

```

```

    }
    else
    {
        portaCom.OpenSerialPort(portNameComboBox.SelectedItem.ToString(),
Convert.ToInt32(comboBoxBaud.SelectedItem));
    }

    portaCom.InitLogger();
    portaCom.StartReading();
    buttonStart.Enabled = false;
    comboBoxBaud.Enabled = false;
    portNameComboBox.Enabled = false;
    buttonStop.Enabled = true;
    buttonSendProg.Enabled = true;
    buttonRun.Enabled = true;
    buttonHalt.Enabled = true;
    btnCanais.Enabled = true;
    buttonUnload.Enabled = true;
    buttonTest.Enabled = true;
    buttonStatus.Enabled = true;
    btnMemoria.Enabled = true;
    btnListagem.Enabled = true;
    buttonCartao.Enabled = true;
    btnRProg.Enabled = true;
    btnCProg.Enabled = true;
    btnCData.Enabled = true;
    btnReset.Enabled = true;
    buttonLimpa.Enabled = true;
    buttonCopia.Enabled = true;
    textBox1.ReadOnly = false;
    textBox1.Enabled = true;
    btnGo.Enabled = true;
    btnStop.Enabled = false;
    labelPortOn.BackColor = System.Drawing.ColorTranslator.FromHtml("#00FF00");
}

private void buttonStop_Click(object sender, EventArgs e)
{
    CloseFichMonit(); //Fecha os ficheiros da monitorização
    if (file != null) file.Close(); //Fecha o ficheiro do Unload se este estiver aberto
    if (portaCom == null) return;
    portaCom.StopReading();
    portaCom.CloseSerialPort();
    portaCom = null;
    buttonStart.Enabled = true;
    comboBoxBaud.Enabled = true;
    portNameComboBox.Enabled = true;
    buttonStop.Enabled = false;
    textBox1.ReadOnly = true;
    buttonSendProg.Enabled = false;
    buttonRun.Enabled = false;
    buttonHalt.Enabled = false;
    btnCanais.Enabled = false;
    buttonUnload.Enabled = false;
    buttonTest.Enabled = false;
    btnMemoria.Enabled = false;
    buttonStatus.Enabled = false;
    btnMemoria.Enabled = false;
    btnListagem.Enabled = false;
    buttonCartao.Enabled = false;
    btnRProg.Enabled = false;
    btnCProg.Enabled = false;
    btnCData.Enabled = false;
    btnReset.Enabled = false;
    buttonLimpa.Enabled = false;
    buttonCopia.Enabled = false;
    btnGo.Enabled = false;
    btnStop.Enabled = false;
    btnMonit = false;
    labelPortOn.BackColor = System.Drawing.ColorTranslator.FromHtml("#3C5F3A");
}

```

```

private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    // Envia um caracter para o buffer.
    try
    {
        portaCom.EscrevePorta(e.KeyChar);
    }
    catch (Exception ex)
    {
        textBox2.Text += "Erro ao tentar escrever para a porta série" + ex.StackTrace;
    }
}

private void buttonSendProg_Click(object sender, EventArgs e)
{
    string mensagem1, mensagem2;
    mensagem1 = "Quer continuar? Vai apagar o programa e os dados da memória do ";
    mensagem1 += "logger e consoante as instruções enviadas poderá apagar o programa e a memória do cartão";
    mensagem2 = "AVISO";
    //Mensagem a pedir a confirmação do envio do programa
    DialogResult dr = MessageBox.Show(mensagem1, mensagem2, MessageBoxButtons.YesNo);
    switch (dr)
    {
        case DialogResult.Yes:
            break;
        case DialogResult.No:
            return;
    }
    //Define a diretoria de default como aquela onde se encontra a correr o programa
    string initialDirectory = System.IO.Path.GetDirectoryName(Application.ExecutablePath);
    openFileDialog1.InitialDirectory = initialDirectory;
    //Abre a janela do browse do ficheiro
    DialogResult result = openFileDialog1.ShowDialog();
    ficheiro = openFileDialog1.FileName;
    //Antes de enviar o programa limpa as schedules existentes e o programa do cartão
    portaCom.EscrevePorta("H");
    portaCom.EscrevePorta("CSCANS");
    portaCom.EscrevePorta("CPRG");
    //Envia o programa para o logger
    if (File.Exists(ficheiro)) portaCom.EnviaProg(ficheiro);
}

private void buttonRun_Click(object sender, EventArgs e)
{
    runProg = new FormRun();
    runProg.texto = "Run";
    runProg.InicializaForm();
    runProg.porta = portaCom;
    runProg.Show();
}

private void buttonHalt_Click(object sender, EventArgs e)
{
    runProg = new FormRun();
    runProg.texto = "Halt";
    runProg.InicializaForm();
    runProg.porta = portaCom;
    runProg.Show();
}

private void buttonUnload_Click(object sender, EventArgs e)
{
    btnMemo = false;
    btnTest = false;
    btnMonit = false;
    unload = new FormUnload();
    unload.porta = portaCom;
    unload.Show();
    RxString = ""; //Limpa o string RxString antes de começar a receber dados
}

```

```

}

private void buttonTest_Click(object sender, EventArgs e)
{
    btnTest = true; //Serve para identificar que o botão TEST foi carregado e por isso
                  // vai guardar os valores, que vão aparecendo na 2ª janela, num ficheiro
    btnMemo = false;
    btnUload = false;
    btnStatus = false;
    btnCns = false;
    btnLista = false;
    btnChart = false;
    btnMonit = false;
    RxString = ""; //Limpa o string RxString antes de começar a receber dados
    strTest = ""; //Limpa o string strTest antes de começar a receber dados
    //portaCom.EscrevePorta("/e/H");
    portaCom.EscrevePorta("TEST");
    Thread.Sleep(4000);
    resultaTest = new FormTest();
    resultaTest.Show();
}

private void btnMemoria_Click(object sender, EventArgs e)
{
    btnMemo = true; //Serve para identificar que o botão Memória foi carregado e
                  //por isso vai guardar as respostas que aparecem na 2ª janela
    btnTest = false;
    btnUload = false;
    btnStatus = false;
    btnCns = false;
    btnChart = false;
    btnLista = false;
    btnMonit = false;
    RxString = ""; //Limpa o string RxString antes de começar a receber dados
    strTest = ""; //Limpa o string strTest antes de começar a receber dados
    //portaCom.EscrevePorta("/e/H");
    portaCom.EscrevePorta("STATUS6");
    Thread.Sleep(250);
    portaCom.EscrevePorta("STATUS7");
    Thread.Sleep(250);
    resultaMemoria = new FormMemoria();
    resultaMemoria.Show();
}

private void buttonStatus_Click(object sender, EventArgs e)
{
    btnStatus = true; //Serve para identificar que o botão Status foi carregado e
                    //por isso vai guardar as respostas que aparecem na 2ª janela
    btnTest = false;
    btnUload = false;
    btnMemo = false;
    btnCns = false;
    btnChart = false;
    btnLista = false;
    btnMonit = false;
    RxString = ""; //Limpa o string RxString antes de começar a receber dados
    strTest = ""; //Limpa o string strTest antes de começar a receber dados
    //portaCom.EscrevePorta("/e/H");
    //GestorPortaSerie.portaSerie.DiscardInBuffer(); //Limpa os buffers de entrada e saída
    //GestorPortaSerie.portaSerie.DiscardOutBuffer();
    portaCom.EscrevePorta("STATUS");
    Thread.Sleep(1000);
    status = new FormStatus();
    status.Show();
}

private void buttonCanais_Click(object sender, EventArgs e)
{
    btnCns = true; //Serve para identificar que o botão Chart foi carregado e
                 //por isso vai guardar as respostas que aparecem na 2ª janela

```

```

        btnStatus = false;
        btnTest = false;
        btnUload = false;
        btnMemo = false;
        btnChart = false;
        btnLista = false;
        btnMonit = false;
        RxString = ""; //Limpa o string RxString antes de começar a receber dados
        strTest = ""; //Limpa o string strTest antes de começar a receber dados
        //portaCom.EscrevePorta("/e/H");
        portaCom.EscrevePorta("STATUS10");
        Thread.Sleep(1000);
        _horarios = new FormSched();
        _horarios.Show();
    }

    private void btnListagem_Click(object sender, EventArgs e)
    {
        //Envia para um ficheiro com o nome "PRG" + (data hora) e extensão "txt" a lista de
        spans/polinómios
        //e as Schedules do programa existente na memória interna e a listagem do programa
        existente no
        //cartão.

        straux = ""; //Limpa os strings straux e RxString antes de começar a receber dados
        RxString = "";

        //mySerialPort.DiscardInBuffer(); //Limpa os buffers de entrada e saída
        //mySerialPort.DiscardOutBuffer();

        //Definição do nome do ficheiro com as listagens dos programas.
        ficheiro = Util.NomeFich("PRG", ".txt", false);
        if (!File.Exists(ficheiro))
        {
            File.Create(ficheiro).Dispose();
        }
        //StreamWriter file = new StreamWriter(ficheiro);
        file = new StreamWriter(ficheiro);
        btnLista = true; //Serve para identificar que o botão Listagens foi carregado
        btnCns = false;
        btnStatus = false;
        btnTest = false;
        btnUload = false;
        btnMemo = false;
        btnChart = false;
        btnMonit = false;
        //portaCom.EscrevePorta("/e/H");
        portaCom.EscrevePorta("STATUS2 STATUS4 STATUS8");
        Thread.Sleep(2000);
    }

    private void buttonCartao_Click(object sender, EventArgs e)
    {
        //btnCartao = true; //Serve para identificar que o botão Status foi carregado e
        //por isso vai guardar as respostas que aparecem na 2ª janela
        btnCns = false;
        btnStatus = false;
        btnTest = false;
        btnUload = false;
        btnMemo = false;
        btnChart = false;
        btnLista = false;
        btnMonit = false;
        string mensagem1, mensagem2;
        mensagem1 = "Este comando formata o cartão de memória, por isso\n";
        mensagem1 += "vai apagar o programa e os dados do cartão.\nQuer continuar?";
        mensagem2 = "AVISO";
        //Mensagem a pedir a confirmação do envio do programa
        DialogResult dr = MessageBox.Show(mensagem1, mensagem2, MessageBoxButtons.YesNo);
        switch (dr)
        {
            case DialogResult.Yes:

```



```

        break;
        case DialogResult.No:
            return;
    }
    portaCom.EscrevePorta("CTEST");
    Thread.Sleep(250);
}

private void btnRPprog_Click(object sender, EventArgs e)
{
    portaCom.EscrevePorta("RUNPROG");
}

private void btnCProg_Click(object sender, EventArgs e)
{
    string mensagem1, mensagem2;
    mensagem1 = "Quer continuar? Vai apagar o programa do cartão";
    mensagem2 = "AVISO";
    //Mensagem a pedir a confirmação do envio do programa
    DialogResult dr = MessageBox.Show(mensagem1, mensagem2, MessageBoxButtons.YesNo);
    switch (dr)
    {
        case DialogResult.Yes:
            break;
        case DialogResult.No:
            return;
    }
    portaCom.EscrevePorta("CPROG");
}

private void btnCData_Click(object sender, EventArgs e)
{
    string mensagem1, mensagem2;
    mensagem1 = "Quer continuar? Vai apagar os dados da memória do cartão";
    mensagem2 = "AVISO";
    //Mensagem a pedir a confirmação do envio do programa
    DialogResult dr = MessageBox.Show(mensagem1, mensagem2, MessageBoxButtons.YesNo);
    switch (dr)
    {
        case DialogResult.Yes:
            break;
        case DialogResult.No:
            return;
    }
    portaCom.EscrevePorta("CDATA");
}

private void btnReset_Click(object sender, EventArgs e)
{
    string mensagem1, mensagem2;
    mensagem1 = "Quer continuar? Vai apagar o programa e os dados da memória interna";
    mensagem2 = "AVISO";
    //Mensagem a pedir a confirmação do envio do programa
    DialogResult dr = MessageBox.Show(mensagem1, mensagem2, MessageBoxButtons.YesNo);
    switch (dr)
    {
        case DialogResult.Yes:
            break;
        case DialogResult.No:
            return;
    }
    portaCom.EscrevePorta("RESET");
    Thread.Sleep(4000);
    //portaCom.EscrevePorta("P17 = 120 P22 = 59 P24=13"); Dão ERRO
    portaCom.EscrevePorta("P14=600 P26=30 P31=1 P39=0 P36=0");
    Thread.Sleep(250);
    portaCom.EscrevePorta("/c/D/e/H/L/n/O/Q/R/S/T/u");
    Thread.Sleep(250);
}
}

```

```

private void buttonLimpa_Click(object sender, EventArgs e)
{
    textBox2.Clear();
}

private void buttonCopia_Click(object sender, EventArgs e)
{
    //Definição do nome do ficheiro com o Unload dos dados. Contem a data e a hora a
    //que é criado e fica na directoria do programa
    ficheiro = Util.NomeFich("TxBBox", ".txt", false);
    if (!File.Exists(ficheiro))
    {
        File.Create(ficheiro).Dispose();
    }
    StreamWriter file = new StreamWriter(ficheiro);
    file.WriteLine(textBox2.Text); //Escreve para o ficheiro o conteúdo da janela inferior
    string mensagem1, mensagem2;
    mensagem1 = "Foi criado o ficheiro:\n " + ficheiro;
    mensagem2 = "INFORMAÇÃO";
    //Mensagem a pedir a confirmação do envio do programa
    DialogResult dr = MessageBox.Show(mensagem1, mensagem2);
    file.Close();
}

private void rchTxBBoxPeri_TextChanged(object sender, EventArgs e)
{
}

private void rchTxBBoxAtras_TextChanged(object sender, EventArgs e)
{
}

private void btnGo_Click(object sender, EventArgs e)
{
    abrFichMonit = true;
    if (gHora == null)
    {
        gHora = new GestorHora();
        gHora.TaNaHora += JaTaNaHora;
    }
    gHora.AtualizaSched(rchTxBBoxPeri.Text, rchTxBBoxAtras.Text);
    if (gHora.aCorrer)
    {
        btnMonit = true;
        btnGo.Enabled = false;
        btnStop.Enabled = true;
        btnStatus = false; //Serve para identificar que o botão Status foi carregado e
        //por isso vai guardar as respostas que aparecem na 2ª janela

        btnTest = false;
        btnUload = false;
        btnMemo = false;
        btnCns = false;
        btnChart = false;
        btnLista = false;
        RxString = ""; //Limpa o string RxString antes de começar a receber dados
        strTest = ""; //Limpa o string strTest antes de começar a receber dados
        portaCom.EscrevePorta("STATUS10"); //Para obter quais os aparelhos existentes em cada
        horário
        Thread.Sleep(1000);
    }
}

private void btnStop_Click(object sender, EventArgs e)
{
    gHora.Stop();
    btnGo.Enabled = true;
}

```

```

        btnStop.Enabled = false;
        CloseFichMonit(); //Fecha os ficheiros da monitorização
    }

    void JaTaNaHora(object s, EventArgs e)
    {
        // Note: this method is called in the thread context, thus we must
        // use Invoke to talk to UI controls. So invoke a method on our
        // thread.
        Invoke(new EventHandler<DataEventArgs>(TaNaHoraSync), new object[] { s, e });
    }

    void TaNaHoraSync(object s, EventArgs e) //Faz o Unload das últimas leituras, para guardar em
    ficheiro
    {
        string strhora = ((DataEventArgs)e).Data;
        DateTime hora, datref;
        TimeSpan dt;
        string dataini, datafim;
        TimeSpan dtini, dtfim;
        datref = new DateTime(1989, 1, 1);
        hora = Convert.ToDateTime(strhora);
        dt = new TimeSpan(0, 0, 1, 0); //(dias, horas, minutos, segundos)
        dtini = hora - datref - dt;
        dtfim = hora - datref + dt;
        dataini = dtini.TotalSeconds.ToString();
        dataini = "(" + dataini + ",0)";
        datafim = dtfim.TotalSeconds.ToString();
        datafim = "(" + datafim + ",0)";
        portaCom.EscrevePorta("UM" + dataini + datafim);
        segundaVez = true;
        //segundaVez = true: quando pela COM é transmitido o unload da leitura
        //segundaVez = false: quando pela COM é transmitida a leitura
    }

    private void textBox2_TextChanged(object sender, EventArgs e)
    {
    }

    #endregion

    #region Auxiliar

    private void CloseFichMonit()
    {
        fichMonit = GestorHora.file;
        int i = -1;
        foreach (StreamWriter monitFile in fichMonit)
        {
            if (monitFile != null) monitFile.Close(); //Fecha os ficheiros da monitorização se
            estiverem abertos
            i++;
            if (contLin[i] == 0)
            {
                File.Delete(GestorHora.ficheiro[i]); //Apaga os ficheiros da monitorização se
                estiverem vazios
            }
            contLin[i] = 0; //Repõe o contador de linhas do ficheiro a zero
        }
    }

    private void DisplayText()
    {
        textBox2.AppendText(RxString);
        string straux1, straux2, mensagem1, mensagem2, cabecalho;
        int iaux0, i, strauxlength;
        string sched, letra;
    }

```

```

DateTime data;
List<string> leitura;

// Se o botão de UNLOAD tiver sido carregado escreve também para um ficheiro *.txt
// A linha seguinte deve-se ao facto de poder não existir a form unload
btnUload = (unload != null ? unload.btnUload : false);
btnChart = (_horarios != null ? _horarios.btnChrt : false);

if (_horarios != null)
    btnChart = _horarios.btnChrt;

if (abrFichMonit)
{
    straux += RxString;
    if (straux.Contains(">:"))
    {
        _dicCanais = Util.separaStrCanais(straux);
        RxString = "";
        straux = "";
        abrFichMonit = false;
        fichMonit = GestorHora.file;
        foreach (var schd in _dicCanais)
        {
            letra = schd.Key;
            cabecalho = "datahora";
            foreach (string strapar in schd.Value)
            {
                cabecalho += "\t" + strapar;
            }
            //Escrita do cabeçalho do ficheiro
            switch (letra)
            {
                case "A":
                    fichMonit[0].WriteLine(cabecalho);
                    _strcanaais[0] = cabecalho;
                    break;
                case "B":
                    fichMonit[1].WriteLine(cabecalho);
                    _strcanaais[1] = cabecalho;
                    break;
                case "C":
                    fichMonit[2].WriteLine(cabecalho);
                    _strcanaais[2] = cabecalho;
                    break;
                case "D":
                    fichMonit[3].WriteLine(cabecalho);
                    _strcanaais[3] = cabecalho;
                    break;
            }
        }
    }
}

if (btnLista)
{
    straux += RxString; //RxString é o string que é recebido na porta série e vai sendo
concatenado
    if (straux.Contains("\r\n")) //A sequência <CR><LF> indica o fim da transmissão
    {
        file.WriteLine(straux); //Escreve a listagem para o ficheiro
        file.Close(); //Fecha o ficheiro
        mensagem1 = "Foi criado o ficheiro:\n " + ficheiro;
        mensagem2 = "INFORMAÇÃO";
        //Mensagem a pedir a confirmação do envio do programa
        DialogResult dr = MessageBox.Show(mensagem1, mensagem2);
        btnLista = false;
        straux = "";
        RxString = "";
    }
}

```

```

if (btnULoad)
{
    file = unload.file;
    straux += RxString; //RxString é o string que é recebido na porta série e vai
                        //straux += strlido; //RxString é o string que é recebido na porta
                        //série e vai sendo colocado na janela inferior.
    if (straux.Contains("\r"))
    {
        int cnt = 0;
        foreach (char c in straux)
        {
            if (c == '\r') cnt++;
        }

        for (i = 0; i < cnt; i++)
        {
            iaux0 = straux.IndexOf("\r");
            strauxlength = straux.Length;
            straux1 = straux.Substring(0, iaux0 + 1);
            if (strauxlength - iaux0 - 2 > 0)
            {
                straux2 = straux.Substring(iaux0 + 1, strauxlength - iaux0 - 1);
            }
            else
            {
                straux2 = "";
            }
            straux = straux2;
            if (!straux1.Contains("::"))
            {
                straux1 = Util.TrataString(straux1);
                file.WriteLine(straux1); //Escreve para o ficheiro os valores
            }
            else
            {
                file.Close(); //Fecha o ficheiro e liberta o botão UNLOAD
                ficheiro = unload.ficheiro;
                mensagem1 = "Foi criado o ficheiro:\n " + ficheiro;
                mensagem2 = "INFORMAÇÃO";
                //Mensagem a pedir a confirmação do envio do programa
                DialogResult dr = MessageBox.Show(mensagem1, mensagem2);
                btnULoad = false;
                unload.btnULoad = btnULoad;
                unload.Close();
            }
        }
    }
}

if (btnTest)
{
    //RxString é o string que é recebido na porta série e aparece na janela inferior.
    strTest += RxString;
    if (strTest.Contains(",23:") || resultaTest.IsDisposed)
    {
        btnTest = false;
        //Preenche a Form resultaTeste
        resultaTest.preencheFormTest(strTest);
    }
}

if (btnMemo)
{
    //RxString é o string que é recebido na porta série e aparece na janela inferior.
    strTest += RxString;
    int cnt = 0;
    foreach (char c in strTest)
    {
        if (c == ':') cnt++;
    }
    if (cnt == 4 || resultaMemoria.IsDisposed)
    {
        btnMemo = false;
        //Preenche a Form resultaMemoria
    }
}

```

```

        resultaMemoria.preencheFormMemoria(strTest);
    }
}

if (btnStatus)
{
    //RxString é o string que é recebido na porta série e aparece na janela inferior.
    strTest += RxString;

    //if (strTest.Split(new[] { "\r\n" }, StringSplitOptions.None).Length >= 9 ||
status.IsDisposed)
    if (strTest.Contains(",9:") || status.IsDisposed)
    {
        btnStatus = false;
        //Preenche a Form resultaTeste
        status.preencheFormStatus(strTest);
    }
}

if (btnCns)
{
    //Determina os canais lidos em cada Schedule
    straux += RxString; //RxString é o string que é recebido na porta série e vai
    if (straux.Contains(">:"))
    {
        _dicCanais = Util.separaStrCanais(straux);
        btnCns = false;
        //Preenche a FormSched
        _horarios.preencheFormSched(_dicCanais);
    }
}

if (btnChart)
{
    straux += RxString; //RxString é o string que é recebido na porta série e vai
    //sendo colocado na janela inferior.
    if (straux.Contains("\r"))
    {
        int cnt = 0;
        foreach (char c in straux)
        {
            if (c == '\r') cnt++;
        }

        for (i = 0; i < cnt; i++)
        {
            iaux0 = straux.IndexOf("\r");
            strauxlength = straux.Length;
            straux1 = straux.Substring(0, iaux0 + 1); //straux1 contem uma linha completa
            if (strauxlength - iaux0 - 2 > 0)
            {
                straux2 = straux.Substring(iaux0 + 1, strauxlength - iaux0 - 1);
            }
            else
            {
                straux2 = "";
            }
            straux = straux2; //Inicializa straux com o que sobrou da construção de straux1
            //Extrai valores fornece a List leituras com os valores lidos na Schedule
            sched, na (data e hora) data
            leitura = Util.ExtraiValores(straux1, out sched, out data);
            //dicCanEsc contem uma key que corresponde à schedule e uma lista de inteiros
            //que corresponde ao índice do aparelho escolhido dentro da schedule
            dicCanEsc = _horarios.dicCanEsc;
            chart = _horarios.chart;
            serieID = _horarios.serieID;
            //Cria um handler de eventos correspondentes a clicar num dos botões do rato
            chart.MouseDown += new MouseEventHandler(_horarios.ButtonDown);
            //Adiciona leituras à chart
            Util.AddXYChart(chart, sched, data, leitura, dicCanEsc, serieID);
        }
    }
}
}

```

```

if (btnMonit & segundaVez)
{
    //A primeira vez corresponde ao envio dos valores medidos para a COM pelo logger e a
    //segunda vez (segundavez = true) ao unload do cartão de memória, com o delay definido
    fichMonit = GestorHora.file;
    //Contadores do número de linhas dos ficheiros
    for (i = 0; i < 4; i++) contLin[i] = GestorHora.contLin[i];

    strMonit += RxString; //RxString é o string que é recebido na porta série e vai
                        //Straux = RxString; //RxString é o string que é recebido na
                        //porta série e vai sendo colocado na janela inferior.
    if (strMonit.Contains("\r"))
    {
        int cnt = 0;
        foreach (char c in strMonit)
        {
            if (c == '\r') cnt++;
        }

        for (i = 0; i < cnt; i++)
        {
            iaux0 = strMonit.IndexOf("\r");
            strauxlength = strMonit.Length;
            straux1 = strMonit.Substring(0, iaux0 + 1);
            if (strauxlength - iaux0 - 2 > 0)
            {
                straux2 = strMonit.Substring(iaux0 + 1, strauxlength - iaux0 - 1);
            }
            else
            {
                straux2 = "";
            }
            strMonit = straux2;
            if (!straux1.Contains("::"))
            {
                straux1 = Util.TrataString(straux1);
                char[] separa = { ';' }; //Separa o string num array de strings
                string[] straux3 = straux1.Split(separa);
                letra = straux3[1];
                int tamanho = letra.Length;
                if (tamanho > 1)
                {
                    {
                        letra = letra.Substring(1, 17);
                    }
                }
                //Separa a escrita das linhas pelos ficheiros correspondentes aos horários
                switch (letra)
                {
                    case "A":
                        straux1 = Regex.Replace(straux1, @";A", "");
                        straux1 = Regex.Replace(straux1, @";", "\t");
                        //Converte a data do formato dd-MM-aaaa para aaaa-MM-dd
                        straux1 = Util.MudaData(straux1);
                        fichMonit[0].WriteLine(straux1); //
                        contLin[0]++;
                        GestorHora.contLin[0] = contLin[0];
                        fichMonit[0].Close();
                        fichMonit[0] = File.AppendText(GestorHora.ficheiro[0]);
                        break;
                    case "B":
                        straux1 = Regex.Replace(straux1, @";B", "");
                        straux1 = Regex.Replace(straux1, @";", "\t");
                        //Converte a data do formato dd-MM-aaaa para aaaa-MM-dd
                        straux1 = Util.MudaData(straux1);
                        fichMonit[1].WriteLine(straux1); //Escreve os valores para ficheiro
                        contLin[1]++;
                        GestorHora.contLin[1] = contLin[1];
                        fichMonit[1].Close();
                        fichMonit[1] = File.AppendText(GestorHora.ficheiro[1]);
                        break;
                    case "C":
                        straux1 = Regex.Replace(straux1, @";C", "");
                        straux1 = Regex.Replace(straux1, @";", "\t");
                        //Converte a data do formato dd-MM-aaaa para aaaa-MM-dd
                        straux1 = Util.MudaData(straux1);
                }
            }
        }
    }
}

```

```

        fichMonit[2].WriteLine(straux1); //Escreve os valores para ficheiro
        contLin[2]++;
        GestorHora.contLin[2] = contLin[2];
        fichMonit[2].Close();
        fichMonit[2] = File.AppendText(GestorHora.ficheiro[2]);
        break;
    case "D":
        straux1 = Regex.Replace(straux1, @";D", "");
        straux1 = Regex.Replace(straux1, @";", "\t");
        //Converte a data do formato dd-MM-aaaa para aaaa-MM-dd
        straux1 = Util.MudaData(straux1);
        fichMonit[3].WriteLine(straux1); //Escreve os valores para ficheiro
        contLin[3]++;
        GestorHora.contLin[3] = contLin[3];
        fichMonit[3].Close();
        fichMonit[3] = File.AppendText(GestorHora.ficheiro[3]);
        break;
    case "data memory empty":
        segundaVez = false;
        break;
    }
}
else
{
    segundaVez = false;
    //Fecha os ficheiros da monitorização e abre novos de hora a hora
    horaAtual = DateTime.Now;
    horaFich = GestorHora.horaFich;
    if (horaFich != horaAtual & horaAtual.Minute == 0)
    {
        CloseFichMonit();
        fichName = GestorHora.ficheiro;
        fichMonit = GestorHora.file;
        GestorHora.AbreFicheiros(fichName, fichMonit, true); //Abre novos
        //ficheiros

        horaFich = DateTime.Now;
        GestorHora.ficheiro = fichName;
        GestorHora.file = fichMonit;
        i = -1;
        foreach (StreamWriter monitFile in fichMonit)
        {
            i++;
            monitFile.WriteLine(Form1.strcanais[i]); //Escreve os cabeçalhos
        }
    }
}
}
}

private void portNameComboBox_SelectedIndexChanged(object sender, EventArgs e)
{
}

private void btnSalvaConf_Click(object sender, EventArgs e)
{
    try
    {
        //Salva a configuração escolhida no ficheiro pref.xml
        pref.BaudRate = Convert.ToInt16(comboBoxBaud.SelectedItem); //Baud rate da porta COM
        pref.NomePorta = Convert.ToString(portNameComboBox.SelectedItem); //nome da porta COM
        pref.Periodo = rchTxtBoxPeri.Text; //Período das leituras da monitorização
        pref.Delay = rchTxtBoxAtras.Text; //Atraso na escrita das leituras para ficheiro
        pref.ModosAuto = cbAuto.Checked; //Check Box para definir a escolha de arranque
        //automático
    }
    catch (Exception ex)
    {
        //Não preencheu corretamente os campos do default
    }
    Config.Save(pref);
}
}

```



```

private void cbAuto_CheckedChanged(object sender, EventArgs e)
{
}

private void setDefaultComboValues()
{
    //Preenche a comboBoxBaud e define o seu valor por default (9600)
    //e também preenche a portNameComboBox definindo ainda o seu valor por default ("COM1")
    string[] NomePortas = new string[10];
    int[] baudrate = new int[5] { 300, 1200, 2400, 4800, 9600 };
    int i, n, m;
    comboBoxBaud.DataSource = baudrate;
    NomePortas = SerialPort.GetPortNames();
    portNameComboBox.DataSource = NomePortas;

    //Carregamento dos valores por default existentes em pref.xml ou
    //em preferences.cs quando não existe pref.xml
    pref = Config.Load();

    //Verificação se a porta existe
    i = -1;
    n = -1;
    foreach(string str in NomePortas)
    {
        i++;
        if (pref.NomePorta == str)
        {
            n = i;
            break;
        }
    }
    if(n > -1)
    {
        portNameComboBox.SelectedIndex = n; //Default portNamecomboBox
    }
    else
    {
        string mensagem1, mensagem2;
        mensagem1 = "A porta série definida por defeito nos ficheiros pref.xml ou
Preferences.cs não existe";
        mensagem2 = "AVISO";
        //Mensagem a pedir a confirmação do envio do programa
        MessageBox.Show(mensagem1, mensagem2, MessageBoxButtons.OK);
    }

    //Verificação se o baud rate é possível
    i = -1;
    m = -1;
    foreach (int veloc in baudrate)
    {
        i++;
        if (pref.BaudRate == veloc)
        {
            m = i;
            break;
        }
    }
    if (m > -1)
    {
        comboBoxBaud.SelectedIndex = m; //Default comboBoxBaud
    }
    else
    {
        string mensagem1, mensagem2;
        mensagem1 = "O baud rate definido por defeito nos ficheiros pref.xml ou Preferences.cs
não existe";
        mensagem2 = "AVISO";
        //Mensagem a pedir a confirmação do envio do programa
        MessageBox.Show(mensagem1, mensagem2, MessageBoxButtons.OK);
    }

    //Se o nome da porta ou o baud rate estiverem errados impede o arranque automático
    if( n < 0 | m < 0)

```

```
    {  
      pref.ModAuto = false;  
    }  
  }  
#endregion  
}
```

ANEXO II
FormLimY.cs


```

//DT500COM - FormLimY.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace DT500COM
{
    public partial class FormLimY : Form
    {
        private string strYMax = "";
        private string strYMin = "";
        private double yMax, yMin;
        private bool _btnclick;

        public FormLimY()
        {
            InitializeComponent();
        }

        public bool btnclick
        {
            get { return _btnclick; }
            set { _btnclick = value; }
        }

        private void txtBoxYMax_TextChanged(object sender, EventArgs e)
        {
        }

        private void txtBoxYMin_TextChanged(object sender, EventArgs e)
        {
        }

        private void FormLimY_Load(object sender, EventArgs e)
        {
            _btnclick = false;
        }

        private void FormLimY_FormClosing(object sender, FormClosingEventArgs e)
        {
            _btnclick = false;
        }

        private void btnLimYOk_Click(object sender, EventArgs e)
        {
            //try
            //{
            strYMax = txtBoxYMax.Text;
            strYMin = txtBoxYMin.Text;
            if (strYMax != "" & strYMin != "")
            {
                yMax = Util.GetDouble(strYMax);
                yMin = Util.GetDouble(strYMin);
                if (yMax <= yMin)
                {
                    MessageBox.Show("Lim. Inf. >= Li. Sup.", "ATENÇÃO");
                    return;
                }
            }
            }
        }
    }
}

```

```
        Util.AlterarEscY(Form1.form1.horarios.chart, yMin, yMax);
    }
    else
    {
        Form1.form1.horarios.chart.ResetAutoValues(); //Atualiza os limites da chart
automaticamente
    }

    //}
    //catch (Exception ex)
    //{
    //    //do nothing
    //}
}
}
}
```

ANEXO III
FormMemoria.cs


```

//DT500COM - FormMemoria.cs
using System;
using System.Windows.Forms;

namespace DT500COM
{
    public partial class FormMemoria : Form
    {
        public FormMemoria()
        {
            InitializeComponent();
        }

        public void preencheFormMemoria(string strTest)
        //Preenchimento da Form Memoria
        {
            int size, i;
            string[] straux3;
            Label[] labels = new Label[4] { lblMiDisp, lblMiOcup, lblMcDisp, lblMcOcup };

            straux3 = Util.separaString(strTest); //Separa o string num array de strings
            size = straux3.Length - 1;

            //Escrita dos valores da memória livre e ocupada
            for (i = 0; i < size; i++)
            {
                string[] s = straux3[i].Split(',');
                labels[2 * i].Text = s[0];
                labels[2 * i + 1].Text = s[1];
            }
        }

        private void FormMemoria_Load(object sender, EventArgs e)
        {
        }
    }
}

```


ANEXO IV
FormRun.cs


```

//DT500COM - FormRun.cs
using System;
using System.Windows.Forms;

namespace DT500COM
{
    public partial class FormRun : Form
    {
        private string comando;
        private string _texto;
        private GestorPortaSerie _porta;

        public FormRun()
        {
            InitializeComponent();
            InicializaForm();
        }

        #region Properties

        public string texto
        {
            get { return _texto; }
            set { _texto = value; }
        }

        public GestorPortaSerie porta
        {
            get { return _porta; }
            set { _porta = value; }
        }

        #endregion

        #region Events

        private void rdBtnTotSched_CheckedChanged(object sender, EventArgs e)
        {
            // Faz executar ou parar todas as Schedules
            InicializaForm();
        }

        private void rdnBtnEscSched_CheckedChanged(object sender, EventArgs e)
        {
            // Faz executar ou parar as Schedules escolhidas
            comando = "";
            chkBoxA.Enabled = true;
            chkBoxB.Enabled = true;
            chkBoxC.Enabled = true;
            chkBoxD.Enabled = true;
            chkBoxStatist.Enabled = false;
            chkBoxAlarms.Enabled = false;
        }

        private void buttonOK_Click(object sender, EventArgs e)
        {
            porta.EscrevePorta(comando);
            Close();
        }

        private void chkBoxA_CheckedChanged(object sender, EventArgs e)
        {
            if (_texto == "Run")
            {
                comando = chkBoxA.Checked ? comando + " GA" : comando.Replace(" GA", "");
            }
            else
            {
            }
        }
    }
}

```

```

        comando = chkBoxA.Checked ? comando + " HA" : comando.Replace(" HA", "");
    }
}

private void chkBoxB_CheckedChanged(object sender, EventArgs e)
{
    if (_texto == "Run")
    {
        comando = chkBoxB.Checked ? comando + " GB" : comando.Replace(" GB", "");
    }
    else
    {
        comando = chkBoxB.Checked ? comando + " HB" : comando.Replace(" HB", "");
    }
}

private void chkBoxC_CheckedChanged(object sender, EventArgs e)
{
    if (_texto == "Run")
    {
        comando = chkBoxC.Checked ? comando + " GC" : comando.Replace(" GC", "");
    }
    else
    {
        comando = chkBoxC.Checked ? comando + " HC" : comando.Replace(" HC", "");
    }
}

private void chkBoxD_CheckedChanged(object sender, EventArgs e)
{
    if (_texto == "Run")
    {
        comando = chkBoxD.Checked ? comando + " GD" : comando.Replace(" GD", "");
    }
    else
    {
        comando = chkBoxD.Checked ? comando + " HD" : comando.Replace(" HD", "");
    }
}

private void chkBoxStatist_CheckedChanged(object sender, EventArgs e)
{
}

private void chkBoxAlarms_CheckedChanged(object sender, EventArgs e)
{
}

private void FormRun_Load(object sender, EventArgs e)
{
}

#endregion

#region Auxiliar

public void InicializaForm()
{
    Text = _texto;
    if (_texto == "Run")
    {
        comando = "G"; //Run - G
    }
    else
}

```

```
{
  comando = "H"; //Halt - H
}
checkBoxA.Enabled = false;
checkBoxA.Checked = false;
checkBoxB.Enabled = false;
checkBoxB.Checked = false;
checkBoxC.Enabled = false;
checkBoxC.Checked = false;
checkBoxD.Enabled = false;
checkBoxD.Checked = false;
checkBoxStatist.Enabled = false;
checkBoxStatist.Checked = false;
checkBoxAlarms.Enabled = false;
checkBoxAlarms.Checked = false;
}

#endregion

}
```


ANEXO V
FormSched.cs


```

//DT500COM - FormSched.cs

using System;
using System.Collections.Generic;
using System.Drawing; //ColorTranslator
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;

namespace DT500COM
{
    public partial class FormSched : Form
    {
        public List<RichTextBox> caixaTexto = new List<RichTextBox>();
        public Point? prevPosition = null;
        public ToolTip tooltip = new ToolTip();
        public FormLimY frmLimY = null;
        private int nCor = -1;
        //Definição das cores das séries da chart
        private static string[] cores = { "#FF0000", "#003FFF", "#FF7F00", "#FFFF00", "#FFFFFF",
"#00FFFF",
        "#FF00FF", "#7F00FF", "#7FFF00", "#7F3200" };
        private int ncores = cores.Length;
        private List<string> canaisEscolhidos;
        private List<int> numaparelhos;
        //private List<int> nChEsc;
        //private FormSched frmSched;
        //private ChartArea chartArea;
        //private bool btnclick = false;
        private Dictionary<string, int> _serieID;
        private Dictionary<string, List<int>> _dicCanEsc;
        private Chart _chart;
        private bool _btnChrt;

        public FormSched()
        {
            InitializeComponent();
        }

        #region Properties

        public Dictionary<string, int> serieID
        {
            get { return _serieID; }
            set { _serieID = value; }
        }

        public Dictionary<string, List<int>> dicCanEsc
        {
            get { return _dicCanEsc; }
            set { _dicCanEsc = value; }
        }

        public Chart chart
        {
            get { return _chart; }
            set { _chart = value; }
        }

        public bool btnChrt
        {
            get { return _btnChrt; }
            set { _btnChrt = value; }
        }

        #endregion

        #region Events

        private void FormSched_Load(object sender, EventArgs e)
        {
            btnCriaChart.Enabled = true;
            btnEscolha.Enabled = true;
        }
    }
}

```

```

}

private void btnEscolha_Click(object sender, EventArgs e)
{
    //int nn;
    //nn = caixaTexto.Count;
    foreach (RichTextBox cxtxt in caixaTexto)
    {
        cxtxt.Focus(); //Vai colocando o "Focus" sequencialmente nas caixas de texto que contêm
        //os nomes dos aparelhos lidos em cada schedule.
        string escolha = cxtxt.SelectedText; //guarda o nome do aparelho que foi "highlighted"
        if (escolha != "")
        {
            rchTxtBxEscolha.Text += (escolha + '\n'); //escreve os nomes dos aparelhos
            escolha = "";
            cxtxt.Select(0, 0); //Faz o "unselect" do aparelho escolhido para se poder
            break;
        }
    }
}

private void rchTxtBxEscolha_TextChanged(object sender, EventArgs e)
{
}

private void btnCriaChart_Click(object sender, EventArgs e)
{
    int cor;
    _chart = new Chart();
    Util.ConfiguraChart(_chart); //Configura a chart (cor, tamanho, posição, tipo de letra,
    etc.)
    _dicCanEsc = new Dictionary<string, List<int>>();
    _serieID = new Dictionary<string, int>(); //Contem par(nome do aparelho ; índice da série
    na chart)
    int i;
    canaisEscolhidos = Util.canaisChart(rchTxtBxEscolha); //Canais escolhidos para a Chart
    int nce = canaisEscolhidos.Count;
    List<int> _nChEsc = new List<int>(); //Lista com o índice dos canais escolhidos
    //canaisTodos = Util.canaisTodos(Form1.form1.dicCanais); //Totalidade dos canais existentes
    Series[] series = new Series[nce];
    i = 0;
    foreach (string cne in canaisEscolhidos) //Cria séries com os nomes dos aparelhos
    escolhidos
    {
        nCor++;
        cor = nCor % ncores;
        series[i] = new Series();
        series[i].Name = cne;
        series[i].ChartType = SeriesChartType.FastLine;
        series[i].XValueType = ChartValueType.DateTime;
        series[i].Color = ColorTranslator.FromHtml(cores[cor]);
        //series[i].XValueType = ChartValueType.Double;
        //series[i].Enabled = true;
        _chart.Series.Add(series[i]);
        _serieID.Add(cne, i);
        i++;
    }
    //Cria um Dictionary com uma key que é a schedule e uma List
    //com o índice do canal escolhido para a chart em cada shchedule
    _dicCanEsc = dicCanEscolhidos(Form1.form1.dicCanais, canaisEscolhidos);
    _btnChrt = true;
    _chart.Visible = true;
    Controls.Add(_chart);
    _chart.Show();
    _chart.GetToolTipText += _chart.GetToolTipText;
    btnCriaChart.Enabled = false;
    btnAltera.Enabled = true;
}

```

```

private void FormSched_FormClosing(object sender, FormClosingEventArgs e)
{
    _btnChrt = false;
}

private void btnAltera_Click(object sender, EventArgs e)
{
    bool criaSerie, delSerie;
    int cor;
    List<string> canaisEscAnt = new List<string>();
    canaisEscAnt = canaisEscolhidos;
    int ncant = canaisEscAnt.Count;
    canaisEscolhidos = Util.canaisChart(rchTxtBxEscolha);
    ///Cria um Dictionary com uma key que é a schedule e uma List
    ///com o índice do canal escolhido para a chart em cada shcedule
    _dicCanEsc = dicCanEscolhidos(Form1.form1.dicCanais, canaisEscolhidos);
    ///Ciclo para acrescentar séries
    foreach (string chn in canaisEscolhidos)
    {
        criaSerie = true;
        foreach (string chnAnt in canaisEscAnt)
        {
            if (chn == chnAnt)
            {
                criaSerie = false;
                break;
            }
        }
        if (criaSerie)
        {
            nCor++;
            cor = nCor % ncores;
            Series serie = new Series();
            //_chart.Series.Remove(serie);
            _chart.Series.Add(serie);
            serie.Name = chn;
            serie.ChartType = SeriesChartType.FastLine;
            serie.XValueType = ChartValueType.DateTime;
            serie.Color = ColorTranslator.FromHtml(cores[cor]);
        }
    } //Fim do ciclo para acrescentar séries

    ///Ciclo para apagar séries
    foreach (string chnAnt in canaisEscAnt)
    {
        delSerie = true;
        foreach (string chn in canaisEscolhidos)
        {
            if (chnAnt == chn)
            {
                delSerie = false;
                break;
            }
        }
        if (delSerie)
        {
            _chart.Series.RemoveAt(_chart.Series.IndexOf(chnAnt));
        }
    } //Fim do ciclo para apagar séries

    _serieID = new Dictionary<string, int>(); //Determina o par(nome do aparelho ; índice da
série na chart)
    foreach (Series serie in _chart.Series)
    {
        _serieID.Add(serie.Name, _chart.Series.IndexOf(serie.Name));
    }
}

public void ButtonDown(object sender, MouseEventArgs mea)
{

```

```

    if (frmLimY == null)
    {
        frmLimY = new FormLimY();
    }
    else
    {
        if (!frmLimY.btnclick) frmLimY = new FormLimY();
    }

    if (mea.Button == MouseButtons.Right)
    { //Check for right button click
        richTextBox1.Text = "X position = " + mea.X.ToString() + "\r\n" +
            "Y position = " + mea.Y.ToString() + "\n" + "\r\n";
    }
    else if (mea.Button == MouseButtons.Left)
    { //Check for Left button click
    }

    if (mea.X < 80 & mea.Y < 250 & !frmLimY.btnclick)
    {
        frmLimY.TopMost = true; //Estas duas linhas são para a frmLimY ficar por cima das
        outras forms
        frmLimY.BringToFront();
        frmLimY.Show();
        frmLimY.btnclick = true;
    }
}

void _chart_MouseMove(object sender, MouseEventArgs e)
{
    //Determina a coordenada do gráfico quando o cursor do rato se encontra
    //a menos de 2 pixel de distância do gráfico.
    var pos = e.Location;
    if (prevPosition.HasValue && pos == prevPosition.Value)
        return;
    tooltip.RemoveAll();
    prevPosition = pos;
    var results = _chart.HitTest(pos.X, pos.Y, false, ChartElementType.DataPoint);
    foreach (var result in results)
    {
        if (result.ChartElementType == ChartElementType.DataPoint)
        {
            var prop = result.Object as DataPoint;
            if (prop != null)
            {
                var pointXPixel = result.ChartArea.AxisX.ValueToPixelPosition(prop.XValue);
                var pointYPixel = result.ChartArea.AxisY.ValueToPixelPosition(prop.YValues[0]);

                // check if the cursor is really close to the point (2 pixels around the point)
                if (Math.Abs(pos.X - pointXPixel) < 2 &&
                    Math.Abs(pos.Y - pointYPixel) < 2)
                {
                    tooltip.Show("X=" + prop.XValue + ", Y=" + prop.YValues[0], this._chart,
                        pos.X, pos.Y - 15);
                }
            }
        }
    }
}

private void _chart_GetToolTipText(object sender, ToolTipEventArgs e)
{
    // Check selected chart element and set tooltip text for it
    switch (e.HitTestResult.ChartElementType)
    {
        case ChartElementType.DataPoint:
            var dataPoint = e.HitTestResult.Series.Points[e.HitTestResult.PointIndex];
            e.Text = string.Format("X:\t{0}\nY:\t{1}", dataPoint.XValue, dataPoint.YValues[0]);
            break;
    }
}
}

```

```

#endregion

#region Auxiliar

public void preencheFormSched(Dictionary<string, List<string>> dicHorarios) //(A,
("TP01", "TP02", .....))
//Preenchimento da FormSched
{
    numaparelhos = new List<int>();
    List<Label> lblSched = new List<Label>();
    int numchaves = dicHorarios.Count;
    int n = -1;
    int nap = 0;
    foreach (var horario in dicHorarios)
    {
        n++;
        nap += horario.Value.Count; //Determina a cumulante do n.º de aparelhos por schedule
        numaparelhos.Add(nap);
        //Construção das textboxes
        RichTextBox caixa = new RichTextBox();
        caixa.AutoSize = false;
        caixa.Size = new System.Drawing.Size(900, 25);
        caixa.Location = new Point(160, 25 + n * 90);
        //caixa.Anchor = AnchorStyles.Top | AnchorStyles.Left;
        caixa.Visible = true;
        Controls.Add(caixa);
        //Construção das labels
        Label lblHorario = new Label();
        lblHorario.AutoSize = false;
        lblHorario.Size = new System.Drawing.Size(120, 20);
        lblHorario.Font = new Font(FontFamily.GenericSansSerif, 10, FontStyle.Bold);
        lblHorario.Location = new Point(30, 25 + n * 90);
        lblHorario.Visible = true;
        lblHorario.Text = "Schedule: " + horario.Key.ToString();
        //lblHorario.Anchor = AnchorStyles.Top | AnchorStyles.Left;
        Controls.Add(lblHorario);
        foreach (string aparelho in horario.Value)
        {
            caixa.Text += (aparelho + " ");
        }
        caixaTexto.Add(caixa);
    }
}

public Dictionary<string, List<int>> dicCanEscolhidos(Dictionary<string,
List<string>> dicTotCan, List<string> chnEsc)
{
    Dictionary<string, List<int>> dicCanEsc = new Dictionary<string, List<int>>();
    int i;
    int nce = canaisEscolhidos.Count;
    //Determinação do n.º de aparelhos escolhidos por shedule
    //O Dictionary dicTotCan é constituído pela letra que identifica a schedule e por
    //uma lista de strings com os nomes dos aparelhos lidos em cada horário.
    //O Dictionary dicCanEsc é constituído pela letra que identifica a schedule e por
    //uma lista de integers com o índice do aparelho escolhido na respetiva schedule.
    //chnEsc é uma List de strings com os nomes dos aparelhos escolhidos para a chart
    foreach (var par in dicTotCan)
    {
        i = -1;
        List<int> lstCanEsc = new List<int>(); //Lista com o índice do canal escolhido em cada
schedule
        foreach (var aprlh in par.Value)
        {
            i++;
            foreach (string aparEsc in chnEsc)
            {
                if (aprlh == aparEsc)
                { lstCanEsc.Add(i); }
            }
        }
        dicCanEsc.Add(par.Key, lstCanEsc);
    }
}

```

```
        return dicCanEsc;
    }
    #endregion
} }
```


ANEXO VI
FormStatus.cs


```

//DT500COM - FormStatus.cs
using System;
using System.Drawing;
using System.Windows.Forms;

namespace DT500COM
{
    public partial class FormStatus : Form
    {
        public FormStatus()
        {
            InitializeComponent();
        }

        private void FormStatus_Load(object sender, EventArgs e)
        {
        }

        public void preencheFormStatus(string strTest)
        //Preenchimento da Form Status
        {
            int size, nlinha, i, n, numlbl;
            string[] straux3;
            string[] s;
            char[] letra = new char[5];
            int[] nltr = new int[5];
            Label[] label1 = new Label[5] { lblSchA, lblSchB, lblSchC, lblSchD, lblSchX };
            Label[] label2 = new Label[12] { lblAddress, lblVersao, lblAlarmAct, lblAlarmInact,
                lblSpan, lblLogging, lblMIFree, lblMISor, lblMCFree, lblMCStor, lblProgFree,
                lblProgStor};

            straux3 = Util.separaString(strTest); //Separa o string num array de strings
            size = straux3.Length - 1;
            numlbl = 0;
            for (nlinha = 0; nlinha < 8; nlinha++)
            {
                switch (nlinha)
                {
                    case 0: //Endereço e Versão
                    case 2: //Alarmes
                    case 5: //Memória interna
                    case 6: //Memória do cartão
                    case 7: //Memória do programa do cartão
                        s = straux3[nlinha].Split(',');
                        label2[numlbl].Text = s[0];
                        label2[numlbl + 1].Text = s[1];
                        numlbl += 2;
                        break;
                    case 1: //Schedules
                        s = straux3[nlinha].Split(',');
                        if (s[0] != "none")
                        {
                            string[] horarios = s[0].Split(' '); // O array string contém as letras dos
                            //horários definidos

                            n = horarios.Length;
                            for (i = 0; i < n; i++)
                            {
                                letra[i] = System.Convert.ToChar(horarios[i]);
                                nltr[i] = Convert.ToInt32(letra[i]);
                                nltr[i] = (nltr[i] < 69 ? nltr[i] - 65 : nltr[i] - 84); //Schedule A-D
                                //e X
                                //Verde
                                label1[nltr[i]].BackColor = ColorTranslator.FromHtml("#00FF00");
                            }
                        }
                        if (s[1] != "none")
                        {
                            string[] horarios = s[1].Split(' '); // O array string contém as letras dos
                            //horários definidos

                            n = horarios.Length;
                            for (i = 0; i < n; i++)

```

```
        {
            letra[i] = System.Convert.ToChar(horarios[i]);
            nltr[i] = Convert.ToInt32(letra[i]);
            nltr[i] = (nltr[i] < 69 ? nltr[i] - 65 : nltr[i] - 84); //Schedule A-D

            label1[nltr[i]].BackColor = ColorTranslator.FromHtml("#FF9600");
        }
    }
    break;
case 3: //Spans e polinómios
    s = straux3[nlinha].Split(',');
    label2[numlbl].Text = s[0];
    numlbl++;
    break;
case 4: //Logging
    s = straux3[nlinha].Split(',');
    s[0] = (s[0] == "1" ? "On" : "Off");
    label2[numlbl].Text = s[0];
    numlbl++;
    break;
}
}
}
}
```

ANEXO VII
FormTest.cs


```

//DT500COM - FormTest.cs
using System;
using System.Drawing;
using System.Windows.Forms;

namespace DT500COM
{
    public partial class FormTest : Form
    {
        public FormTest()
        {
            InitializeComponent();
        }

        private void FormTest_Load(object sender, EventArgs e)
        {

        }

        public void preencheFormTest(string strTest)
        //Preenchimento da Form TESTE
        {
            int size, i;
            string[] straux3;
            Label[] labels = new Label[36] {lblDatataker, lblVos, lblVfo, lblFc, lblCMRR,
                lblVos3, lblTos, lblIos, lblIbia, lblIbat, lblVbat, lblVosa, lblVosm, lblVosMM,
                lblVosR, lblVosd, lblIcs1, lblIcs2, label137, label18, label112, label116, label118,
                label20, label22, label24, label26, label28, label29, label30, label31, label32,
                label33, label34, label35, label36};

            straux3 = Util.separaString(strTest); //Separa o string num array de strings
            size = straux3.Length - 1;

            //definição dos valores e luz da 1ª linha
            string[] s = straux3[0].Split(',');
            labels[0].Text = s[0];
            labels[18].Text = s[1];

            //Escrita do resultado global
            if (straux3[size - 1] == "1")
            {
                lblResultado.Text = "PASSOU NO TESTE";
                lblResultado.BackColor = ColorTranslator.FromHtml("#3C6339");
                lblResultado.ForeColor = ColorTranslator.FromHtml("#00FF00");
            }
            else
            {
                lblResultado.Text = "CHUMBOU NO TESTE";
                lblResultado.BackColor = ColorTranslator.FromHtml("#822321");
                lblResultado.ForeColor = ColorTranslator.FromHtml("#FF0000");
            }
        }

        //definição dos valores e luz da 2ª linha e seguintes
        for (i = 1; i < size - 1; i++)
        {
            s = straux3[i].Split(',');
            labels[i].Text = s[0];
            if (s[1] == "1")
            {
                labels[i + size - 1].BackColor = ColorTranslator.FromHtml("#00FF00");
            }
            else
            {
                labels[i + size - 1].BackColor = ColorTranslator.FromHtml("#FF0000");
            }
        }
    }
}

```


ANEXO VIII
FormUnload.cs


```

//DT500COM - FormUnload.cs
using System;
using System.Windows.Forms;
using System.IO;
using System.Threading;
using System.Globalization;

namespace DT500COM
{
    public partial class FormUnload : Form
    {
        private string cmd1, cmd2, cmd3, cmd4, comando;
        private string datahora, strdathorini, strdathorfim, _ficheiro;
        private DateTime datahoraIniMI, datahoraIniMC, datahoraFimMI, datahoraFimMC;
        private DateTime datahoraIni, datahoraFim;
        private bool _btnULoad;
        private GestorPortaSerie _porta;
        private StreamWriter _file;

        public FormUnload()
        {
            InitializeComponent();
            cmd1 = "UM";
            cmd2 = "";
            cmd3 = "(BEGIN)";
            cmd4 = "(END)";
            _btnULoad = false;
        }

        #region Properties

        public string ficheiro
        {
            get { return _ficheiro; }
            set { _ficheiro = value; }
        }

        public bool btnULoad
        {
            get { return _btnULoad; }
            set { _btnULoad = value; }
        }

        public GestorPortaSerie porta
        {
            get { return _porta; }
            set { _porta = value; }
        }

        public StreamWriter file
        {
            get { return _file; }
            set { _file = value; }
        }

        #endregion

        #region Events

        private void rdbtnMI_CheckedChanged(object sender, EventArgs e)
        {
            cmd1 = "UI";
        }

        private void rdbtnMC_CheckedChanged(object sender, EventArgs e)
        {
            cmd1 = "UM";
        }

        private void rdbtnMIMC_CheckedChanged(object sender, EventArgs e)
        {
            cmd1 = "U";
        }
    }
}

```

```

}

private void rdbtnTot_CheckedChanged(object sender, EventArgs e)
{
    cmd2 = "";
}

private void rdbtnA_CheckedChanged(object sender, EventArgs e)
{
    cmd2 = "A";
}

private void txtBoxDatIni_TextChanged(object sender, EventArgs e)
{
}

private void FormUnload_Load(object sender, EventArgs e)
{
}

private void rdbtnB_CheckedChanged(object sender, EventArgs e)
{
    cmd2 = "B";
}

private void rdbtnC_CheckedChanged(object sender, EventArgs e)
{
    cmd2 = "C";
}

private void rdbtnD_CheckedChanged(object sender, EventArgs e)
{
    cmd2 = "D";
}

private void rdbtnIni_CheckedChanged(object sender, EventArgs e)
{
    cmd3 = "(BEGIN)";
}

private void rdbtnFim_CheckedChanged(object sender, EventArgs e)
{
    cmd4 = "(END)";
}

private void rdbtnDatIni_CheckedChanged(object sender, EventArgs e)
{
    DatasLimite(); //Determina as datas limite dos dados armazenados
}

private void rdbtnDatFim_CheckedChanged(object sender, EventArgs e)
{
    DatasLimite(); //Determina as datas limite dos dados armazenados
}

private void rdbtnX_CheckedChanged(object sender, EventArgs e)
{
    cmd2 = "X";
}

private void BtnOK_Click(object sender, EventArgs e)
{
    string dataini, datafim;
    DateTime datref, dtinicial, dtfinal;
    TimeSpan dtini, dtfim;

    datref = new DateTime(1989, 1, 1);

    cmd3 = "(BEGIN)";
    if (rdbtnDatIni.Checked)
    {

```

```

        dtinicial = DateTime.ParseExact(txtBoxDatIni.Text, "HH:mm:ss dd-MM-yyyy",
CultureInfo.InvariantCulture);
        dtini = dtinicial - datref;
        dataini = dtini.TotalSeconds.ToString();
        dataini = "(" + dataini + ",0)";
        cmd3 = dataini;
    }

    cmd4 = "(END)";
    if (rdbtnDatFim.Checked)
    {
CultureInfo.InvariantCulture);
        dtfinal = DateTime.ParseExact(txtBoxDatFim.Text, "HH:mm:ss dd-MM-yyyy",
CultureInfo.InvariantCulture);
        dtfim = dtfinal - datref;
        datafim = dtfim.TotalSeconds.ToString();
        datafim = "(" + datafim + ",0)";
        cmd4 = datafim;
    }

    comando = cmd1 + cmd2 + cmd3 + cmd4;

    //Definição do nome do ficheiro com o Unload dos dados. Contem a data e a hora a
    //que é criado e fica na directoria do programa
    _ficheiro = Util.NomeFich("DT", ".csv");
    if (!File.Exists(_ficheiro))
    {
        File.Create(_ficheiro).Dispose();
    }

    //StreamWriter file = new StreamWriter(ficheiro);
    _file = new StreamWriter(_ficheiro);

    _btnULoad = true; //Serve para identificar que o botão UNLOAD foi carregado e
    //por isso vai guardar os valores, que vão aparecendo na 2ª janela, num
ficheiro
    _porta.EscrevePorta(comando);
}

#endregion

#region Auxiliar

private void DatasLimite()
{
    _porta.EscrevePorta("STATUS12");
    Thread.Sleep(250);
    datahora = _porta.LePorta();
    Util.DataHora(datahora, out datahoraIniMI, out datahoraFimMI);

    _porta.EscrevePorta("STATUS13");
    Thread.Sleep(250);
    datahora = _porta.LePorta();
    Util.DataHora(datahora, out datahoraIniMC, out datahoraFimMC);

    if (rdbtnMI.Checked)
    {
        datahoraIni = datahoraIniMI;
        datahoraFim = datahoraFimMI;
    }
    if (rdbtnMC.Checked)
    {
        datahoraIni = datahoraIniMC;
        datahoraFim = datahoraFimMC;
    }
    if (rdbtnMIMC.Checked)
    {
        datahoraIni = (datahoraIniMI < datahoraIniMC ? datahoraIniMI : datahoraIniMC);
        datahoraFim = (datahoraFimMI > datahoraFimMC ? datahoraFimMI : datahoraFimMC);
    }
    strdathorini = datahoraIni.ToString("HH:mm:ss dd/MM/yyyy");
    txtBoxDatIni.Text = strdathorini;
    strdathorfim = datahoraFim.ToString("HH:mm:ss dd/MM/yyyy");
    txtBoxDatFim.Text = strdathorfim;
}
}

```

```
    #endregion  
  }  
}
```

ANEXO IX
GestHora.cs


```

//DT500COM - GestHora.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Globalization;
using System.Threading;
using System.Windows.Forms;
using System.IO;

namespace DT500COM
{
    public class GestorHora
    {
        private Thread thrdHora;
        private DateTime periodo, delay, horaArranque;
        private int difperiodo;
        private bool _aCorrer = false;
        private bool primeira;
        private static List<string> _ficheiro = new List<string>();
        private static List<StreamWriter> _file = new List<StreamWriter>();
        private static int[] _contLin = { 0, 0, 0, 0 };
        private static DateTime _horaFich;

        public GestorHora()
        {
        }

        #region Properties

        public static DateTime horaFich
        {
            get { return _horaFich; }
            set { _horaFich = value; }
        }

        public static List<string> ficheiro
        {
            get { return _ficheiro; }
            set { _ficheiro = value; }
        }

        public static List<StreamWriter> file
        {
            get { return _file; }
            set { _file = value; }
        }

        public bool aCorrer
        {
            get { return _aCorrer; }
            set { _aCorrer = value; }
        }

        public static int[] contLin
        {
            get { return _contLin; }
            set { _contLin = value; }
        }

        #endregion

        #region Events

        // Estes eventos são desencadeados na background thread
        public event EventHandler<DataEventArgs> TaNaHora;

        #endregion

        #region Auxiliar

```

```

public void AtualizaSched(string periodoStr, string atrasoStr)
{
    try
    {
        periodo = DateTime.ParseExact(periodoStr, "HH:mm", CultureInfo.InvariantCulture);
        delay = DateTime.ParseExact(atrasoStr, "mm:ss", CultureInfo.InvariantCulture);
    }
    catch (FormatException)
    {
        MessageBox.Show("Existe pelo menos um dado com o formato errado", "Atenção");
        _aCorrer = false;
        return;
    }
    difperido = periodo.Hour * 60 + periodo.Minute;
    if (1440 % difperido != 0 | difperido < 2)
    {
        MessageBox.Show("O tempo escolhido para a periodocidade deve ser" +
            " sub-múltiplo do dia (24 h = 1440 min) e maior ou igual a 2 min", "Atenção");
        return;
    }
    if ((delay.Minute * 60 + delay.Second) < 30)
    {
        MessageBox.Show("O atraso na escrita para ficheiro deve ser" +
            " maior que 30 segundos", "Atenção");
        return;
    }
    if (difperido * 60 - delay.Minute * 60 - delay.Second < 30)
    {
        MessageBox.Show("O atraso na escrita para ficheiro deve ser, pelo menos," +
            " 30 segundos inferior à periodicidade", "Atenção");
        return;
    }
    Start();
    _aCorrer = true;
}

public void Start()
{
    thrdHora = new Thread(RunMethod);
    thrdHora.Start();
    horaArranque = DateTime.Now;
    primeira = true;
}

public void Stop()
{
    if (thrdHora != null) thrdHora.Abort();
}

public static void AbreFicheiros(List<string> nomeFich, List<StreamWriter> nomeStream, bool
automatico)
{
    //Definição dos nomes dos ficheiros com os dados da monitorização. Contêm a data e a hora a
    //que são criados e ficam na diretoria do programa
    string[] nomes = { "DTA@", "DTB@", "DTC@", "DTD@" };
    //Limpeza das listas eventualmente existentes reeferentes aos ficheiros e respetivos
streams
    int lim = nomeStream.Count;
    for (int j = lim - 1; j >= 0; j--)
    {
        nomeStream.RemoveAt(j);
        nomeFich.RemoveAt(j);
    }
    //Abertura dos ficheiros
    int i = -1;
    foreach (string nome in nomes)
    {
        i++;
        nomeFich.Add(Util.NomeFich(nome, ".txt", automatico));
        if (!File.Exists(nomeFich[i]))
        {

```

```

        File.Create(nomeFich[i]).Dispose();
    }
    nomeStream.Add(new StreamWriter(nomeFich[i]));
}

private void RunMethod()
{
    string strhora = "";
    TimeSpan difTmpArranque;
    int difminutos, desfini, folga, tempo, tSeguinte, tAgora;
    double desfini2 = 0;
    double dtAgora, dtempo;
    string strhorant = "";
    DateTime horaAtual;
    folga = 60; //folga de tempo no arranque em segundos para não fazer uma leitura,
    //por exemplo, às 17:13:58 e a seguinte às 17:14:00 (passados só 2 s)
    while (true)
    {
        horaAtual = DateTime.Now;
        strhora = horaAtual.ToString("HH:mm,dd/MM/yyyy", DateTimeFormatInfo.InvariantInfo);
        //DateTime.TryParse(strhora, out hora);
        difminutos = horaAtual.Hour * 60 + horaAtual.Minute;
        desfini = horaAtual.Second; //Determina o segundo em que arrancou para só
        //desencadear acções na mudança exata do minuto (HH:mm:00)
        difTmpArranque = horaAtual - horaArranque;
        desfini2 = difTmpArranque.TotalSeconds - desfini + difperiodo * 60;

        if (primeira) //Fase do arranque
        {
            //Abertura dos ficheiros relativos à monitorização, caso em que o último argumento
            //é "true"
            AbreFicheiros(_ficheiro, _file, true);
            _horaFich = horaAtual;
            primeira = false;
            if (difminutos % difperiodo == 0)
            {
                tempo = 0; //Se estiver em cima da hora não pára a thread
            }
            else //Pára a thread o tempo que falta até à 1ª leitura, menos 2 segundos
            {
                tAgora = horaAtual.Hour * 60 + horaAtual.Minute;
                dtAgora = tAgora + (double)horaAtual.Second / 60;
                tSeguinte = (tAgora / difperiodo + 1) * difperiodo;
                dtempo = (tSeguinte - dtAgora) * 60000 - 2000;
                tempo = (dtempo > 0 ? (int)dtempo : 0);
            }
            Thread.Sleep(tempo);
        }
        else //Fases seguintes
        {
            if (difminutos % difperiodo == 0 & strhorant != strhora)
            {
                strhorant = strhora;
                if (TaNaHora != null & desfini2 >= folga)
                {
                    Thread.Sleep(delay.Minute * 60000 + delay.Second * 1000 - 2000);
                    TaNaHora(this, new DataEventArgs(strhora));
                }
            }
        }
    }
}

#endregion
}
}

```


ANEXO X
GestPortaSerie.cs


```

//DT500COM - GestPortaSerie.cs
using System;
using System.Threading;
using System.IO.Ports;
using System.Globalization;
using System.IO;

public class GestorPortaSerie
{
    private Thread thrdPortaSerie;
    private static SerialPort mySerialPort;

    public GestorPortaSerie()
    {
        thrdPortaSerie = new Thread(RunMethod);
        mySerialPort = new SerialPort();
    }

    public void OpenSerialPort(string portName, int baudRate)
    {
        mySerialPort.BaudRate = baudRate;
        mySerialPort.PortName = portName;
        //mySerialPort.BaudRate = 9600;
        //mySerialPort.PortName = "COM1";
        mySerialPort.Parity = Parity.None;
        mySerialPort.StopBits = StopBits.One;
        mySerialPort.DataBits = 8;
        mySerialPort.Handshake = Handshake.None;
        mySerialPort.ReadTimeout = 10000;
        mySerialPort.WriteTimeout = 10000;
        //Abertura da porta série
        mySerialPort.Open();
        mySerialPort.DiscardInBuffer(); //Limpa os buffers de entrada e saída
        mySerialPort.DiscardOutBuffer();
    }

    public void CloseSerialPort()
    {
        StopReading();
        if (mySerialPort.IsOpen)
        {
            mySerialPort.DiscardInBuffer(); //Limpa os buffers de entrada e saída
            Thread.Sleep(250);
            mySerialPort.DiscardOutBuffer();
            mySerialPort.Close();
        }
    }

    public bool SerialPortIsOpen()
    {
        return mySerialPort.IsOpen;
    }

    public void InitLogger()
    {
        mySerialPort.WriteLine("/e/H/R" + '\r'); // \e: O logger não ecoa os comandos
        Thread.Sleep(250);
        mySerialPort.WriteLine("P31=1 P39=0" + '\r'); // Formato: dd/mm/yyyy; hh:mm:ss
        Thread.Sleep(250);
    }

    public void StartReading()
    {
        thrdPortaSerie.Start();
    }

    public void StopReading()
    {

```

```

        if (thrdPortaSerie != null)
            thrdPortaSerie.Abort();
    }

    // Estes eventos são desencadeados na background thread
    public event EventHandler<DataEventArgs> DataReceived;

    public void EscrevePorta(char caracter)
    {
        char[] buff = new char[1];
        buff[0] = caracter;
        mySerialPort.Write(buff, 0, 1);
    }

    public void EscrevePorta(string str)
    {
        mySerialPort.WriteLine(str + '\r');
    }

    public string LePorta()
    {
        return mySerialPort.ReadLine();
    }

    public void EnviaProg(string ficheiro)
    {
        //Envia o programa para o logger
        string line, data, hora;
        int conta = 0;
        // Garante o formato desejado para a data e hora
        mySerialPort.WriteLine("" + '\r'); //Envia um Return para acordar o logger
        mySerialPort.WriteLine("P31=1" + '\r'); //Formta a data para dd/mm/yyyy
        Thread.Sleep(300);
        mySerialPort.WriteLine("P39=0" + '\r'); //Formata a hora para hh:mm:ss
        Thread.Sleep(300);
        //Acerta a data e a hora do logger pela data e pela hora do computador
        data = DateTime.Now.ToString("dd/MM/yyyy", DateTimeFormatInfo.InvariantInfo);
        hora = DateTime.Now.ToString("HH:mm:ss", DateTimeFormatInfo.InvariantInfo);
        mySerialPort.WriteLine("D=" + data + '\r');
        Thread.Sleep(300);
        mySerialPort.WriteLine("T=" + hora + '\r');
        Thread.Sleep(300);
        //Lê o ficheiro do programa enviando-o linha a linha para o logger.
        StreamReader fich = new StreamReader(@ficheiro);
        while ((line = fich.ReadLine()) != null)
        {
            conta += 1;
            mySerialPort.WriteLine(line + '\r');
            switch (line)
            {
                case "RESET":
                    Thread.Sleep(5000);
                    break;
                case "CLEAR":
                    Thread.Sleep(1000);
                    break;
                default:
                    Thread.Sleep(300);
                    break;
            }
        }
        fich.Close();
    }

    private void RunMethod()
    {
        string linant = "";
        string line = "";
    }

```



```
while (mySerialPort.IsOpen)
{
    linant = line;
    line = mySerialPort.ReadExisting();
    //Controla se há atividade na porta verificando se a linha anterior foi modificada
    if (DataReceived != null && line != linant) DataReceived(this, new DataEventArgs(line));
}
}
```


ANEXO XI
Preferences.cs


```

//DT500COM - Preferences.cs

using System;
using System.IO;
using System.Xml.Serialization;

namespace DT500COM
{
    public static class Config
    {
        //Para usar System.Configuration.ConfigurationManager é necessário adicionar uma assembly
        // reference de System.Configuration.
        //para tal, no Solution Explorer, faz-se right click em References e seleciona-se Add
        //reference. Depois seleciona-se a Assemblies tab e escolhe-se em Framework marca-se a
        //checkbox System.configuration. No ficheiro App.config foi adicionada uma chave (<add key
        //="prefFile" value=".\\pref.xml" />)com a definição do nome e da localização do ficheiro de
        //configuração.

        private static string cfg_path =
System.Configuration.ConfigurationManager.AppSettings["prefFile"];

        public static void Save(Preferences pref)
        {
            XmlSerializer mySerializer = new XmlSerializer(pref.GetType());
            StreamWriter myWriter = new StreamWriter(cfg_path);
            mySerializer.Serialize(myWriter, pref);
            myWriter.Close();
        }

        public static Preferences Load()
        {
            if (File.Exists(cfg_path))
            {
                XmlSerializer mySerializer = new XmlSerializer(typeof(Preferences));
                FileStream myFileStream = new FileStream(cfg_path, FileMode.Open);
                try
                {
                    Preferences pref = (Preferences)mySerializer.Deserialize(myFileStream);
                    myFileStream.Close();
                    return pref;
                }
                catch (Exception e)
                {
                    //Quando não há ficheiro de preferencias...load defaults.
                }
            }
            return new Preferences();
        }
    }

    [Serializable]
    public class Preferences
    {
        public Preferences()
        {
        }

        //Preferencias quando não existe o ficheiro de preferencias (pref.xml)
        private int _baudrate = 9600;
        private string _nomePorta = "COM1";
        private string _periodo = "00:15";
        private string _delay = "01:00";
        private bool _modoAuto = false;

        public int BaudRate
        {
            get { return _baudrate; }
            set { _baudrate = value; }
        }

        public string NomePorta
        {

```

```
    get { return _nomePorta; }
    set { _nomePorta = value; }
}

public string Período
{
    get { return _período; }
    set { _período = value; }
}

public string Delay
{
    get { return _delay; }
    set { _delay = value; }
}

public bool ModoAuto
{
    get { return _modoAuto; }
    set { _modoAuto = value; }
}
}
```

ANEXO XII
Program.cs


```
//DT500COM - Program.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace DT500COM
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```


ANEXO XIII
SerialThread.cs


```

//DT500COM - SerialThread.cs
using System;
using System.Threading;
using System.IO.Ports;

public class SerialReaderThread
{
    private Thread thrdPortaSerie;
    private string _portName;
    private int _baudRate;
    public SerialPort mySerialPort;

    public char _character;

    public char character
    {
        get { return _character; }
        set { _character = value; }
    }

    public SerialReaderThread(string portName, int baudRate)
    {
        _portName = portName;
        _baudRate = baudRate;
        thrdPortaSerie = new Thread(RunMethod);
    }

    public void Start() { thrdPortaSerie.Start(); }

    public void Stop()
    {
    }

    // Estes eventos são desencadeados na background thread
    public event EventHandler<DataEventArgs> DataReceived;
    public event EventHandler<EventArgs> PortaLigada;

    private bool closed = false;
    public void Close()
    {
        closed = true;
        if (mySerialPort.IsOpen)
        {
            mySerialPort.DiscardInBuffer(); //Limpa os buffers de entrada e saída
            Thread.Sleep(250);
            mySerialPort.DiscardOutBuffer();
            mySerialPort.Close();
        }
    }

    public void EscrevePorta(char caracter)
    {
        // Se a porta estiver fechada não envia o comando.
        if (!mySerialPort.IsOpen) return;

        // If the port is Open, declare a char[] array with one element.

        char[] buff = new char[1];

        // Load element 0 with the key character.
        //buff[0] = e.KeyChar;
        buff[0] = caracter;

        // Send the one character buffer.
        mySerialPort.Write(buff, 0, 1);
    }

    private void RunMethod()
    {
        string linant = "";
    }
}

```

```

string line = "";

//Parâmetros de configuração da porta série
//SerialPort mySerialPort = new SerialPort();
mySerialPort = new SerialPort();

mySerialPort.BaudRate = _baudRate;
mySerialPort.PortName = _portName;
mySerialPort.Parity = Parity.None;
mySerialPort.StopBits = StopBits.One;
mySerialPort.DataBits = 8;
mySerialPort.Handshake = Handshake.None;
mySerialPort.ReadTimeout = 10000;
mySerialPort.WriteTimeout = 10000;

//Abertura da porta série
mySerialPort.Open();

mySerialPort.DiscardInBuffer(); //Limpa os buffers de entrada e saída
Thread.Sleep(250);
mySerialPort.DiscardOutBuffer();
mySerialPort.WriteLine("" + '\r'); //Envia um Return para acordar o logger
Thread.Sleep(250);
mySerialPort.WriteLine("/e" + '\r'); // \e: O logger não ecoa os comandos
Thread.Sleep(250);

PortaLigada(this,null);

while (!closed && mySerialPort.IsOpen)
{
    linant = line;
    line = mySerialPort.ReadExisting();
    //string line = mySerialPort.ReadLine();
    //line = "x";

    //Controla se há atividade na porta verificando se a linha anterior foi modificada
    if (DataReceived != null && line != linant)
        DataReceived(this, new DataEventArgs(line));
}
}

public class DataEventArgs : EventArgs
{
    public string Data { get; private set; }

    public DataEventArgs(string data) { Data = data; }
}

```

ANEXO XIV
Util.cs


```

//DT500COM - Util.cs
using System;
using System.Globalization; //Para obter a data e hora
using System.Windows.Forms; //Para abrir uma OpenFileDialog
using System.IO;
using System.Text.RegularExpressions;
using System.Collections.Generic;
using System.Windows.Forms.DataVisualization.Charting;
using System.Drawing;

namespace DT500COM
{
    public static class Util
    {
        private static ChartArea chartArea1;

        public static Dictionary<string, List<string>> separaStrCanais(string strTest)
        {
            //Faz a separação (parse) do string de resposta ao comando "STATUS10"
            //para determinar os nomes das grandezas associadas às leituras efetuadas
            //A coleção dicCanais é do tipo "Dictionary" e é composta por um string com a letra
            //que identifica a schedule e uma "List" com os nomes dos aparelhos lidos nessa
            schedule
            int size, i, conta1, lenght;
            string chave = "";
            Dictionary<string, List<string>> dicCanais = new Dictionary<string, List<string>>();
            List<string> lstCanais = new List<string>();
            char[] separa = { ',' }; //Separa o string num array de strings
            string[] straux = strTest.Split(separa);
            size = straux.Length - 1;
            conta1 = -1;
            i = 0;
            while (i < size)
            {
                if (straux[i].Substring(0, 1) == "<" && straux[i].Length == 2)
                {
                    if (conta1 > -1)
                    {
                        dicCanais.Add(chave, lstCanais);
                    }
                    chave = straux[i].Substring(1, 1);
                    conta1++;
                    i++;
                    lstCanais = new List<string>();
                }
                //Esta proteção existe para só passar pelo "if" seguinte quando straux[i] tiver
                mais de 1 letra
                if (straux[i].Length > 1)
                {
                    if (straux[i].Substring(0, 2) == "<\")
                    {
                        lenght = straux[i].Length;
                        lstCanais.Add(straux[i].Substring(2, lenght - 3));
                    }
                }
                i++;
            }
            dicCanais.Add(chave, lstCanais);
            return dicCanais;
        }

        public static double GetDouble(string s) //Converte string em double independentemente
            //do separador dos decimais ser "." ou ","
        {
            double d;

            var formatinfo = new NumberFormatInfo();

            formatinfo.NumberDecimalSeparator = ".";

            if (double.TryParse(s, NumberStyles.Float, formatinfo, out d))
            {
                return d;
            }
        }
    }
}

```

```

    }

    formatinfo.NumberDecimalSeparator = ",";

    if (double.TryParse(s, NumberStyles.Float, formatinfo, out d))
    {
        return d;
    }

    throw new SystemException(string.Format("Formato do limite da escala errado '{0}'",
s));
}

public static void AddXYChart(Chart chart, string sched, DateTime data,
List<string> leitura, Dictionary<string, List<int>> dicCanEsc, Dictionary<string,
int> serieID)
{
    double y;
    int i, ns;
    string nomeapar;
    foreach (var par in dicCanEsc)
    {
        if (par.Key == sched)
        {
            foreach (int n in par.Value)
            {
                i = -1;
                foreach (string leit in leitura)
                {
                    i++;
                    if (i == n)
                    {
                        //Nome do aparelho correspondente à leitura
                        nomeapar = Form1.form1.dicCanais[sched][i];
                        //Identifica o n.º da série da Chart a partir do nome do
aparelho

                        ns = serieID[nomeapar];
                        //y = Convert.ToDouble(leit);
                        //Não permite ter "." ou "," a separar os decimais
                        y = GetDouble(leit);
                        chart.Series[ns].Points.AddXY(data, y); //Adiciona pontos à
chart
                    }
                }
            }
        }
    }
    //if (chart.Series[j].Points.Count() > 100 ) chart.Series[j].Points.RemoveAt(0);
    //Limita o n.º de pontos da chart
    //chartArea.AxisX.Minimum = 0;
    //chartArea.AxisX.Maximum = 50;
    chart.ResetAutoValues(); //Atualiza os parâmetros da chart
}

public static List<string> canaisTodos(Dictionary<string, List<string>> dicCanais)
{
    List<string> totCanais = new List<string>();

    foreach (var horario in dicCanais)
    {
        foreach (string aparelho in horario.Value)
        {
            totCanais.Add(aparelho);
        }
    }
    return totCanais;
}

```

```

public static void AlteraEscY(Chart chart, double yMin, double yMax)
{
    //Altera os valores máximo e mínimo da escala das ordenadas.
    chartArea1.AxisY.Minimum = yMin;
    chartArea1.AxisY.Maximum = yMax;
}

public static void ConfiguraChart(Chart chart)
{
    //Cria e configura uma chart onde são representados valores medidos para aparelhos
    escolhidos.
    chart.Location = new Point(10, 420);
    chart.Size = new Size(1050, 330);
    chart.BackColor = ColorTranslator.FromHtml("#000000");
    var chartArea = new ChartArea();
    chartArea1 = chartArea; //Para se poder utilizar chartArea no método
    "AlteraEscY(chart, yMin, yMax)" desta classe
    chartArea.AxisX.LabelStyle.Format = "yyyy-MM-dd\\nhh:mm:ss";
    chartArea.BackColor = ColorTranslator.FromHtml("#003000");
    chartArea.AxisX.MajorGrid.LineColor = ColorTranslator.FromHtml("#008000");
    chartArea.AxisY.MajorGrid.LineColor = ColorTranslator.FromHtml("#008000");
    chartArea.AxisX.MajorGrid.LineDashStyle = ChartDashStyle.Dot;
    chartArea.AxisY.MajorGrid.LineDashStyle = ChartDashStyle.Dot;
    chartArea.AxisX.LabelStyle.Font = new Font("Arial Narrow", 8);
    chartArea.AxisY.LabelStyle.Font = new Font("Arial Narrow", 8);
    chartArea.AxisX.LabelStyle.ForeColor = ColorTranslator.FromHtml("#00FF00");
    chartArea.AxisY.LabelStyle.ForeColor = ColorTranslator.FromHtml("#00FF00");
    chart.Legends.Add(new Legend("Apar"));
    chart.Legends["Apar"].Position.Auto = true;
    chart.Legends["Apar"].Font = new Font("Arial Narrow", 8);
    chart.Legends["Apar"].BackColor = ColorTranslator.FromHtml("#003000");
    chart.Legends["Apar"].ForeColor = ColorTranslator.FromHtml("#00FF00");
    //chartArea.AxisX.Minimum = 0;
    //chartArea.AxisX.Maximum = 50;
    chart.ChartAreas.Add(chartArea);

    ///As seis linhas seguintes destinam-se a permitir fazer zoom dos eixos X e Y
    //chart.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;
    //chart.ChartAreas[0].CursorY.IsUserSelectionEnabled = true;
    //chart.ChartAreas[0].AxisX.ScaleView.Zoomable = true;
    //chart.ChartAreas[0].AxisY.ScaleView.Zoomable = true;
    //chart.ChartAreas[0].CursorX.AutoScroll = true;
    //chart.ChartAreas[0].CursorY.AutoScroll = true;
}

public static List<string> canaisChart(RichTextBox rchTxtBxEscolha)
{
    //Faz a separação (parse) do string existente na janela que contem os canais
    //escolhidos para incluir na chart e coloca os nomes dos canais escolhidos numa List
    de strings.
    int size, i;
    List<string> lstCanais = new List<string>();
    string strCanais = rchTxtBxEscolha.Text;
    char[] separa = { '\n' }; //Separa o string num array de strings
    string[] straux = strCanais.Split(separa);
    size = straux.Length - 1;
    i = 0;
    while (i < size)
    {
        if (straux[i] != "")
        {
            straux[i] = Regex.Replace(straux[i], @"\s\n?", "");
            lstCanais.Add(straux[i]);
        }
        i++;
    }
    return lstCanais;
}

public static string[] separaString(string strTest)
{

```

```

int size, i, fim, local;
char[] separa = { '\r' }; //Separa o string num array de strings
string[] straux3 = strTest.Split(separa);

size = straux3.Length - 1;
for (i = 0; i < size; i++)
{
    straux3[i] = Regex.Replace(straux3[i], @"\n?", ""); //Substitui os LF por ""
    fim = straux3[i].Length;
    local = straux3[i].IndexOf(":") + 1; //Localiza a 1ª ocorrência de ":"
    straux3[i] = straux3[i].Substring(local, fim - local - 1); //Conserva apenas os
2 últimos valores
}
return straux3;
}

public static DateTime ConvStringSec(string strsec)
{
    // strsec corresponde à diferença para 1989, em segundos.
    double seconds, date;
    DateTime loggerdate, datref;
    seconds = Convert.ToDouble(strsec) / 3600 / 24; //Conversão de segundos em dias
    datref = new DateTime(1989, 1, 1);
    date = Convert.ToSingle(datref.ToOADate()); //Conversão da data de referência para
double
seconds += date; //soma da data de referência à data do logger
loggerdate = DateTime.FromOADate(seconds); //Conversão de double para datetime
return loggerdate;
}

public static void DataHora(string straux, out DateTime datahoraini, out DateTime
datahorafim)
{
    //Trata os strings que resultam do comando UNLOAD
    string[] strsec = new string[2];
    int fim, ini;
    straux = Regex.Replace(straux, @"\r\n?|\n", ""); //Substitui os CR e os LF por ""

    fim = straux.Length;
    ini = straux.IndexOf(":") + 1; //Localiza a 1ª ocorrência de ":"
    straux = straux.Substring(ini, fim - ini - 1); //Conserva apenas os 2 últimos
valores

    char[] separa = { ',' }; //Separa o string num array de strings
    string[] straux1 = straux.Split(separa);
    datahoraini = ConvStringSec(straux1[0]);
    datahorafim = ConvStringSec(straux1[1]);
}

public static List<string> ExtraiValores(string straux1, out string sched, out DateTime
loggerdate)
{
    //Trata os strings que contêm as leituras que são escritos na TextBox
    //straux1 - string com a linha de leituras
    //sched - letra da schedule
    //loggerdate - time stamp das leituras
    string strsec;
    string[] saux1;
    string saux2;
    double seconds, date;
    int saux1length, i, leitsize;
    DateTime datref;
    List<string> leitura = new List<string>();
    sched = ""; //Inicialização de dois argumentos?????
    loggerdate = new DateTime(1989, 1, 1);
    straux1 = Regex.Replace(straux1, @"\r\n?|\n", ""); //Substitui os CR e os LF por ""
    saux1 = straux1.Split(',');
    if (saux1[0] == "D") //"D" é a 1ª letra de uma mensagem tipo "Data" (Msg. com
leituras)
    {
        strsec = saux1[2];
    }
}

```

```

        seconds = Convert.ToDouble(strsec) / 3600 / 24; //Conversão de segundos em dias
        dateref = new DateTime(1989, 1, 1);
        date = Convert.ToSingle(dateref.ToOADate()); //Conversão da data de referência
para double
        seconds += date; //soma da data de referência à data do logger
        loggerdate = DateTime.FromOADate(seconds); //Conversão de double para datetime
        saux1length = saux1.Length;
        saux2 = saux1[3]; //String com 3 caracteres em que o último é a schedule
        sched = saux2.Substring(2, 1);
        leitsize = saux1length - 5; //Os últimos elementos do string contêm as leituras
        for (i = 0; i < leitsize; i++) //e são adicionados à String List
        {
            saux1[i + 5] = Regex.Replace(saux1[i + 5], @":", ""); //Retira os ":";
            leitura.Add(saux1[i + 5]);
        }
    }
    return leitura;
}

public static string TrataString(string straux1)
{
    //Trata os strings que resultam do comando UNLOAD
    string strsec;
    string[] saux1;
    string[] saux2;
    double seconds, date;
    int saux2length, i;
    DateTime loggerdate, dateref;
    straux1 = Regex.Replace(straux1, @"\r\n?\n", ""); //Substitui os CR e os LF por ""
                                                    //strsec = straux1.Substring(4, 9);
                                                    //Diferença para 1989, em segundos.

    saux1 = straux1.Split(':');
    saux2 = saux1[0].Split(',');
    strsec = saux2[2];
    seconds = Convert.ToDouble(strsec) / 3600 / 24; //Conversão de segundos em dias
    dateref = new DateTime(1989, 1, 1);
    date = Convert.ToSingle(dateref.ToOADate()); //Conversão da data de referência para
double
    seconds += date; //soma da data de referência à data do logger
    loggerdate = DateTime.FromOADate(seconds); //Conversão de double para datetime
    saux2 = saux1[1].Split(',');
    saux2length = saux2.Length;
    straux1 = saux2[0];
    for (i = 2; i < saux2length; i++)
    {
        straux1 += ";" + saux2[i];
    }
    straux1 = Regex.Replace(straux1, @":", ""); //Retira os ":" do fim da linha
    straux1 = loggerdate + ";" + straux1;
    return straux1;
}

public static string NomeFich(string prefixo, string extensao, bool automatico)
{
    //Define o nome do ficheiro onde guarda os dados do UNLOAD Para os ficheiros
    // da monitorização automatico = true e no nome não tem minutos e segundos
    string ficheiro;
    string data, hora, d, M, a, h, m, s;
    //Definição do nome do ficheiro para o Unload dos dados. O nome começa por "DT" e
segue-se a data e
    //a hora a que é criado. O ficheiro é criado na mesma diretoria do programa
    data = DateTime.Now.ToString("dd/MM/yyyy", DateTimeFormatInfo.InvariantInfo);
    hora = DateTime.Now.ToString("HH:mm:ss", DateTimeFormatInfo.InvariantInfo);
    d = data.Substring(0, 2);
    M = data.Substring(3, 2);
    a = data.Substring(6, 4);
    h = hora.Substring(0, 2);
    m = hora.Substring(3, 2);
    s = hora.Substring(6, 2);
    string dir = Path.GetDirectoryName(Application.ExecutablePath);
    if (automatico)
    {

```

```

        ficheiro = dir + '\\\' + prefixo + a + M + d + h + extensao;
    }
    else
    {
        ficheiro = dir + '\\\' + prefixo + a + M + d + h + m + s + extensao;
    }
    return ficheiro;
}

public static string MudaData(string straux1)
{
    ///Muda a data que vem do logger no formato dd-MM-aaaa para aaaa-MM-dd
    string strDataOld, strDataNew, straux;
    int fim = straux1.Length;
    DateTime dataOld;
    strDataOld= straux1.Substring(0, 10);
    dataOld = DateTime.ParseExact(strDataOld, "d", null);
    strDataNew = dataOld.ToString("yyy-MM-dd");
    straux=straux1.Substring(10, fim-10);
    straux1 = strDataNew + straux;
    return straux1;
}

}
}
}

```

