

Adiciona o caminho

```
% Ajustar para o local do projeto
cd('C:\Users\cleyton\Documents\EI\AppColetaAPT')
addpath('./Analysers')

% Os dois primeiros exemplos são testes de conceito.
```

Conexões não implementadas

Ainda não temos nenhuma conexão diferente de TCP.

Reserva para em caso no futuro seja desejável outras formas de conexão.

E para ajudar a desacoplar e permitir sobrecarga de métodos.

```
try
    Analyser.connGPIO('1234')
catch exception
    disp(exception.message)
end
```

Conexão GPIB não implementada.

Execução com o simulador SA2500PC

Alocação dinâmica ainda não implementada (perfumaria prevista):

Caso só passe o IP, teríamos uma opção de auto discovery:

```
% Auto discovery
try
    Analyser.connTCP("localhost");
catch exception
    disp(exception.message)
end
```

Autodiscovery não implementado

```
% Ref. portas a varrer:
% 5025 - Keysight e R&S
% 5555 - R&S EB500
% 9001 - Anritsu
% 34835 - Tektronix
```

Instrumento simulado (Download em [SA2500PC](#)).

Conectado e respondendo à sua identificação:

```
disp('Propriedades:')
```

Propriedades:

```
d = Analyser.connTCP("localhost", 34835)
```

```
d =  
dictionary (string → string) with 6 entries:
```

```
"Factory" → "TEKTRONIX"  
"model"   → "SA2500PC"  
"serial"  → "B000000"  
"version" → "7.050"  
"ip"      → "localhost"  
"port"    → "34835"
```

Instância dinâmica.

Cada fabricante deve ter, na pasta 'Analysers', sua classe como mesmo nome de sua superclasse (prop:Factory), e cada especificidade de um certo modelo (prop:model) deve estar com o mesmo nome, o que permite escalar e isolar os componentes em uma interface unificada, e granularizada para o serviço esperado.

O 'Analyser' deve conter todos os comandos genéricos da SCPI (Standard Commands for Programmable Instruments) e IEEE 488.2 Common Commands, este último inicia por um asterisco..

Com base na IDN que o instrumento responde, a instância sempre herda todos os comandos do 'Analyser'.

No caso, o "is a" está representado no classdef como "classdef TEKTRONIX < Analyser", ou seja, Um Tektronix é um analisador, e o SP2500 é um Tektronx ("classdef SA2500PC < TEKTRONIX")

Neste caso herda os comuns e sobrecarrega os comandos do Tektronix, e os específicos do modelo SA2500PC:

```
disp('Instancia Classes:')
```

Instancia Classes:

```
obj = Analyser.instance(d);
```

Base de comando(TEKTRONIX), modelo (SA2500PC).

O simulador fecha quando recebe o comando de reset, então só fingimos o reset para evitar isso (na implementação específica do modelo). O que ainda oportuna outros casos de uso em casos diferentes, como o EB500 que precisa abrir uma porta específica para receber stream UDP.

Um instrumento real não terá o sufixo PC e portanto automaticamente não herdará esse método, o que possibilita compartilhar comandos e especificar no modelo o que for diferente nele.

Aplicando uma sobrecarga no modelo SA2500PC com a resposta da classe:

```
obj.scpIReset;
```

Simulando um "SCPI Reset" para o modelo SA2500PC.

Comandos gerais de inicialização:

```
obj.startUp()
```

Start Ok.

Teste geral de conectividade (SCPI):

```
obj.ping()
```

Resposta IDN recebida:
TEKTRONIX,SA2500PC,B000000,7.050

Obtém parâmetros do objeto:

```
disp('getSpan:')
```

getSpan:

```
disp(obj.getSpan())
```

10000

```
disp('Parâmetros:')
```

Parâmetros:

```
obj.getParms()
```

ans =
dictionary (string → string) with 10 entries:

```
"Function"    → "NORM"  
"AVGConunt"  → "20"  
"Detection"  → "POS"  
"Power"      → "DBM"  
"FStart"     → "99995000"  
"FStop"      → "100005000"
```

```
"ResAuto"    ? "1"  
"Res"        ? "10"  
"InputGain"  ? "1"  
"Att"        ? "50"
```

Operação

Aqui perguntamos se o parâmetro corresponde ao que foi solicitado (assert). E verificamos as respostas no simulador em tempo real:

```
disp('Pausas para observar o comportamento no simulador:')
```

Pausas para observar o comportamento no simulador:

```
pause(5) % Como o simulador não vai resetar,  
         % temos um tempo para ajustes de teste.  
  
obj.setFreq(120000000)  
obj.setSpan(50000)  
assert(str2double(obj.getSpan) == 50000, 'O Span não foi ajustado.')  
obj.setRes(2000)  
assert(str2double(obj.getRes) == 2000, 'O Span não foi ajustado.')  
  
pause(1)  
obj.setFreq(100000000)  
obj.setSpan(10000)  
  
% Isso gera um alerta na janela do simulador,  
% mas não retorna erro.  
try  
    assert(str2double(obj.getRes()) == 2000, 'A Resolução foi alterada  
automaticamente.')  
catch exception  
    disp('Resolução alterada automaticamente para:')  
    disp(obj.getRes())  
end
```

Resolução alterada automaticamente para:
1000

```
pause(1)
disp('Ajuste em faixa larga')
```

Ajuste em faixa larga

```
obj.setFreq(88000000, 108000000)
obj.setRes(20000) % Warning: Data out of range
try
    assert(str2double(obj.getSpan) == 10000, 'O Span não foi ajustado.')
catch
    disp('Se atribuir o span depois gera erro "out of range"')
end
```

Se atribuir o span depois gera erro "out of range"

```
pause(1)
% Observar o simulador com Spectrum o RBW com Auto em verde
obj.setRes('Auto')
obj.setFreq(88000000, 108000000)

pause(1)
% Escolhe uma portadora para análise
obj.setFreq(100000000)
obj.setSpan(10000)
assert(str2double(obj.getSpan) == 10000, 'O Span não foi ajustado.')

% Cuidado! Os pisos não correspondem aos de um instrumento real!

%obj.preAmp('On')
pause(1)

% Para observar abrir o menu Spectrum -> More -> Ampl
for a = 5:1.5:50
    obj.setAtt(a)

    % O pré desativa sozinho com att acima de 15dB
    % Mesmo expressamente solicitado, sem erro:
    obj.preAmp('On')
end

obj.getTraceData()
```

```
ans =
'#42004
UÂÂÂ³  ÂÂÂ8ÂR  ÂøÚÂµ@ÂÂÎÂÏÿÂ@±Âý4ÂÑÂD  Âf  Â[UÂÂÂ_Â~  Â1ÐÂ!1Â<SÂ'ÂfÂøÂÂq"
```

```
disp("Pronto")
```

Pronto