```
% 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
%     Analysers.Analyser.connGPIB('1234')
% catch exception
%     disp(exception.message)
% end
```

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
%     Analysers.Analyser.connTCP("localhost");
% catch exception
%     disp(exception.message)
% end
%
% % 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:')
% callTCP = Analysers.Analyser.connTCP("localhost", 34835) % Simulador
callTCP = Analysers.Analyser.connTCP("192.168.48.2", 34835)
```

```
callTCP =
  dictionary (string  string) with 6 entries:

    "Factory"  "TEKTRONIX"
    "model"    "SA2500"
    "serial"   "B040211"
    "version"  "7.050"
```

```
"ip"      "192.168.48.2"
"port"    "34835"
```

Instância dinâmica.

Cada fabricante deve ter, na pasta 'Analysers', sua classe como mesmo nome de sua superclasse (prop:Factory), e cada especifidade de um certo modelo (prop:model) deve estar com o mesmo nome, o que permite escalonar 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 = Analysers.Analyser.instance(callTCP);
```

```
    "Analyer: Base de comando do fabricante"    "TEKTRONIX"
```

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 compartilar comandos e especificar no modelo o que for diferente nele.

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

```
obj.scpiReset;
```

```
Analyer.scpiReset: Criando nova conexão TCP.
```

Comandos gerais de inicialização:

```
obj.startUp()
```

```
TEKTRONIX: Start Ok.
```

```

```

Teste geral de conectividade (SCPI):

```
obj.ping()
```

```
Analyser.ping: Mesma conexão
Analyser.ping: Resposta IDN recebida:
TEKTRONIX,SA2500,B040211,7.050
```

```

```

Obtém parâmetros do objeto:

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

```
getSpan:
```

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

```
200000000
```

```
disp('Parâmetros:')
```

```
Parâmetros:
```

```
obj.getParms()
```

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

    "Function"  "NORM"
    "AVGCount"  "20"
    "Detection" "POS"
    "UnitPower" "DBM"
    "FStart"    "759000000"
    "FStop"     "959000000"
    "ResAuto"   "1"
    "Res"       "200000"
    "InputGain" "0"
    "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:')
```

```
Pausas para observar o comportamento:
```

```
% Esse bloco try é para liberar a conexão em caso de erro.
try

    %pause(5) % tempo para ajuste manual de teste.
```

```
    % % Testes
    % 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)

    % disp('Nova conexão simulada pelo ping (caso não responda):')
    % obj.conn = [];

    % Isso gera um alerta na janela do simulador,
    % mas não retorna erro.
    % try
    %     RBW = obj.getRes();
    %     assert(str2double(RBW) == 2000, 'A Resolução foi alterada
automaticamente.')
    % catch exception
    %     disp('Resolução alterada automaticamente para:')
    %     disp(RBW)
    % end

    pause(1)
```

```
disp('Amostra em faixa larga')
```

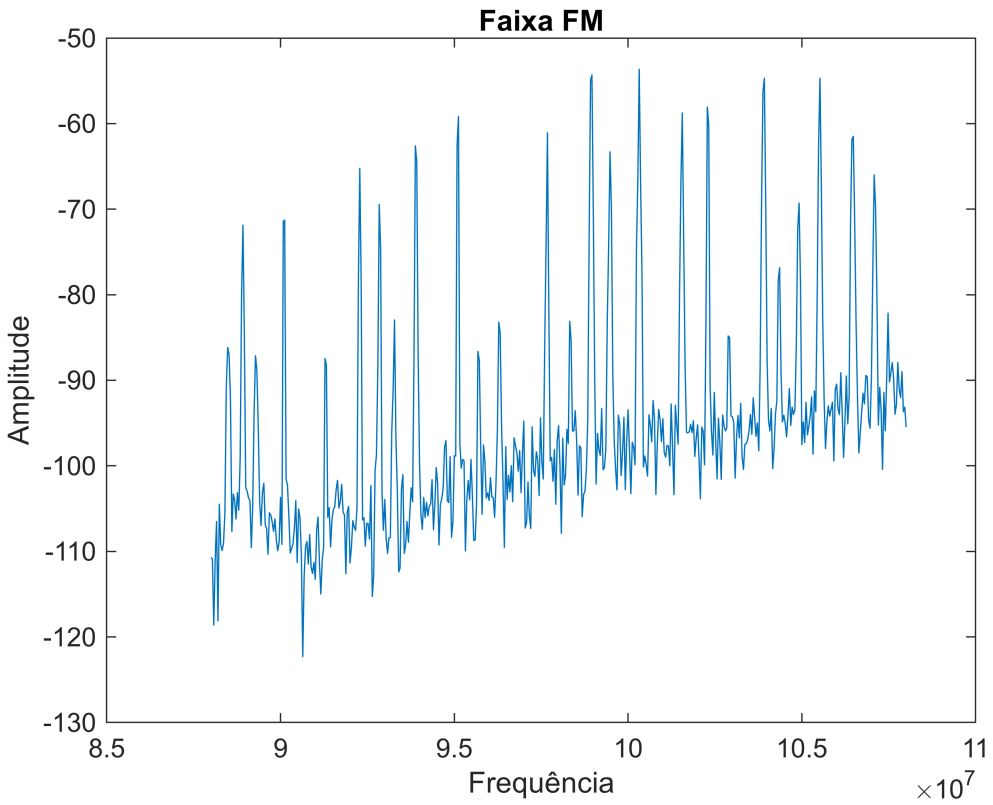
```
Amostra em faixa larga
```

```
obj.setFreq(88000000, 108000000)
obj.setRes(20000) % Warning: Data out of range

wtrace = obj.getTrace(1);

plot(wtrace.freq,wtrace.value,"DisplayName","value");

% Add xlabel, ylabel, title, and legend
xlabel("Frequência");
ylabel("Amplitude");
title("Faixa FM");
```




```
legend('off');
% drawnow;

% try
%     assert(str2double(obj.getSpan) == 10000, 'O Span não foi ajustado.')
% catch
%     disp('Se atribuir o span depois gera erro "out of range"')
% end

% pause(1)
% % Observar que o com Spectrum o RBW Auto em verde
```

```
% obj.setRes('Auto')
% obj.setFreq(88000000, 108000000)

% Para observar a faixa em auto level
pause(2)

% Escolhe uma portadora FM para análise
freq = 100300000;

obj.setSpan(500000)
obj.setFreq(freq)
assert(str2double(obj.getSpan) == 500000, 'O Span não foi ajustado.')

%obj.preAmp('On')

% % Teste dos níveis de atenuação (comentado para poupar tempo)
% % Para observar abrir o menu Spectrum -> More -> Ampl
% for attstep = 5:5:50
%     obj.setAtt(attstep);
%     pause(0.5)
%
%     % O pré desativa sozinho com att acima de 15dB
%     % Mesmo expressamente solicitado, sem erro:
%     obj.preAmp('On')
% end
```

```
trace = obj.getTrace(1);

disp('Trace:')
```

```
Trace:
```

```
% Só as 5 primeiras linhas
disp(trace (1:5,:));
```

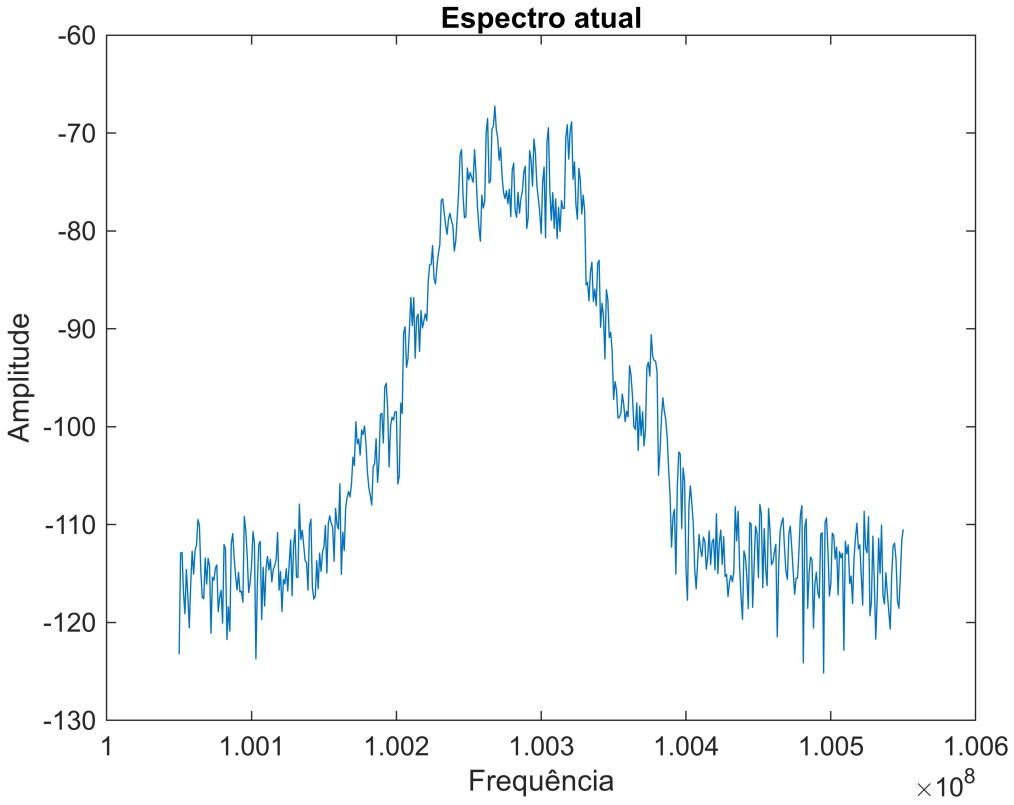
```
      freq         value
    _________    _______

    1.0005e+08    -123.24
    1.0005e+08     -112.9
    1.0005e+08    -112.84
    1.0005e+08    -116.94
    1.0005e+08    -119.11
```

```
% Plota o Trace
```

```
    plot(trace.freq,trace.value,"DisplayName","value");

    % Add xlabel, ylabel, title, and legend
    xlabel("Frequência");
    ylabel("Amplitude");
    title("Espectro atual");
```




```
    legend('off');
```

```
    trcs = fcn.getTracesFromUnit(obj, 30);
```

```
ans =
"Trace 1"
ans =
"Trace 2"
ans =
"Trace 3"
ans =
"Trace 4"
ans =
"Trace 5"
ans =
"Trace 6"
ans =
"Trace 7"
ans =
"Trace 8"
ans =
"Trace 9"
```

```
ans =
"Trace 10"
ans =
"Trace 11"
ans =
"Trace 12"
ans =
"Trace 13"
ans =
"Trace 14"
ans =
"Trace 15"
ans =
"Trace 16"
ans =
"Trace 17"
ans =
"Trace 18"
Analyser: Aguardando sincronismo (1/2): ...
:INITiate:IMMediate;
ans =
"Trace 19"
ans =
"Trace 20"
ans =
"Trace 21"
ans =
"Trace 22"
ans =
"Trace 23"
ans =
"Trace 24"
ans =
"Trace 25"
ans =
"Trace 26"
ans =
"Trace 27"
ans =
"Trace 28"
ans =
"Trace 29"
ans =
"Trace 30"
```

```
    % Verificação adicional.
    if isnan(trcs)
        warning('Trace com NaN')
    end

    disp("Total size:")
```

```
Total size:
```

```
    size(trcs)
```

```
ans = 1×2
    30   501
```

```
    disp(trcs)
```

```
 -119.4926 -114.6828 -113.1192 -114.9349 -115.4292 -115.4454 -113.9077 -112.6733 -119.2177 -109.0816 -112.4341 -109.
 -111.4987 -110.8948 -115.1000 -110.1077 -109.5124 -106.9940 -109.7721 -111.2331 -117.7304 -111.8626 -116.4611 -119.
 -119.5689 -116.8747 -114.0741 -115.3207 -114.1772 -111.5213 -117.0267 -119.8506 -115.9105 -113.2713 -113.8599 -111.
 -113.5794 -116.3129 -114.6740 -119.7829 -112.1491 -107.9356 -118.4173 -117.5620 -114.0354 -109.8775 -115.0151 -113.
 -112.5764 -107.8158 -110.9635 -118.8086 -114.1495 -117.2906 -114.0411 -118.4323 -112.9609 -116.5555 -115.2132 -110.
 -116.2544 -114.3846 -116.0148 -114.6130 -117.4338 -114.4172 -117.2119 -108.9600 -117.4793 -111.0176 -110.2710 -109.
 -122.6898 -121.4326 -121.5023 -113.1945 -114.4917 -108.5172 -107.4863 -113.0321 -112.5567 -112.2992 -118.4694 -114.
 -112.2954 -109.4424 -114.6661 -111.8420 -115.5746 -110.7409 -111.3975 -112.5707 -112.7221 -115.2003 -117.7977 -113.
 -120.6512 -117.8977 -115.6349 -118.9864 -116.9046 -115.7353 -114.9317 -116.7252 -115.8769 -111.2021 -111.8747 -120.
 -113.0703 -109.8640 -109.1631 -108.3859 -108.3667 -115.3054 -111.5262 -113.2381 -114.5837 -113.3851 -112.3429 -122.
 -108.2763 -108.5948 -109.4366 -112.5255 -117.2200 -116.0909 -120.0968 -119.3957 -109.7212 -111.4272 -116.2352 -107.
 -113.8264 -114.5399 -115.8423 -120.7296 -116.4086 -118.1519 -118.0824 -112.6461 -112.1921 -113.0592 -110.4344 -114.
```

```
    % Um tempo para o analisador respirar
    % Porque eventualmente ele não responde
    pause(0.2)

    freq = 100300200;
    obj.setFreq(freq)
    disp('Leitura do marcador:')
```

```
Leitura do marcador:
```

```
    nivel = obj.getMarker(freq, 1);
    fprintf('Em %i MHz o nível é: %f\n', freq, nivel);
```

```
Em 100300200 MHz o nível é: -72.348274
```

```
    if isnan(nivel)
        warning('Nível não retornado.');
    end

catch exception
    % Encerra ativamente a conexão
    % Para liberar a porta em caso de erro
    obj.disconnect();
    error('TestTektronix: %s', exception.identifier);
end

% Se tudo der certo, encerra a conexão.
obj.disconnect()


disp("Pronto")
```

```
Pronto
```