

```
% 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
```

Analyser: Conexão GPIB não implementada.

Execução com a ERMxSC02

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
```

Analyser: Autodiscovery não implementado

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

Conectado e respondendo à sua identificação:

```
% FSL SC
callTCP = Analysers.Analyser.connTCP("10.48.10.241", 5025)
```

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

    "Factory" ☐ "Rohde_Schwarz"
    "model" ☐ "FSL_6"
    "serial" ☐ "100711/006"
    "version" ☐ "2.30"
```

```
"ip"      ? "10.48.10.241"
"port"    ? "5025"
```

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'.

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

```
Instancia Classes:
```

```
obj = Analysers.Analyser.instance(callTCP);
```

```
Analyser: Base de comando(Rohde_Schwarz), modelo (FSL_6).
```

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.

```
obj.scpReset;
```

Comandos gerais de inicialização:

```
obj.startUp()
```

```
R&S: Start Ok.
```

Teste geral de conectividade (SCPI):

```
obj.ping()
```

```
Analyser.ping: Criando conexão
Analyser.ping: Resposta IDN recebida:
Rohde&Schwarz,FSL-6,100711/006,2.30
```

Obtém parâmetros do objeto:

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

```
getSpan:
```

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

```
6000000000
```

```
% A implementar.  
disp('Parâmetros:')
```

```
Parâmetros:
```

```
obj.getParms()
```

```
ans =  
dictionary (string × string) with 16 entries:
```

```
"Trace1Mode"  × "WRIT"  
"SweepCount"  × "0"  
"Function"    × "APE"  
"UnitPower"   × "DBM"  
"FStart"      × "0"  
"FStop"       × "6000000000"  
"SweepPoints" × "501"  
"ResAuto"     × "1"  
"Res"         × "3000000"  
"InputGain"   × "0"  
"AttAuto"     × "1"  
"Att"         × "0"  
"SeewpTime"   × "0.01"  
"FStartMin"   × "0"  
"FStopMax"    × "6000000000"  
"VBW"         × "10000000"
```

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:

```
pause(5)
```

```
obj.setFreq(120000000)
obj.setSpan(50000)
assert(str2double(obj.getSpan) == 50000, 'O Span não foi ajustado.')
obj.setRes(2000)

assert(str2double(obj.getRes) == 3000, 'O Span não foi ajustado.')

pause(1)
obj.setFreq(100000000)
obj.setSpan(10000)

disp('Nova conexão simulada pelo ping (caso não responda):')
```

Nova conexão simulada pelo ping (caso não responda):

```
obj.conn = [];

try
    assert(str2double(obj.getRes()) == 2000, 'A Resolução foi alterada
automaticamente.')
catch exception
    disp('Resolução alterada automaticamente para:')
    disp(obj.getRes())
end
```

Analyzer.getCMD: Criando nova conexão.
Resolução alterada automaticamente para:
3000

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

Ajuste em faixa larga

```
obj.setFreq(88000000, 108000000)
obj.setRes(20000)
try
    assert(str2double(obj.getSpan) == 10000, 'O Span não foi ajustado.')
catch
    disp('Observar o que acontece se setar o span depois.')
end
```

Observar o que acontece se setar o span depois.

```
pause(1)
```

```

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.')

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

% Range de 0 a 30, step 5.
for attstep = 0:5:30
    obj.setAtt(attstep);
    pause(0.5)
    obj.preAmp('On')
end

% TODO: Obtém o traço
trace = obj.getTrace(1);
disp('Trace:')

```

Trace:

```

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

```

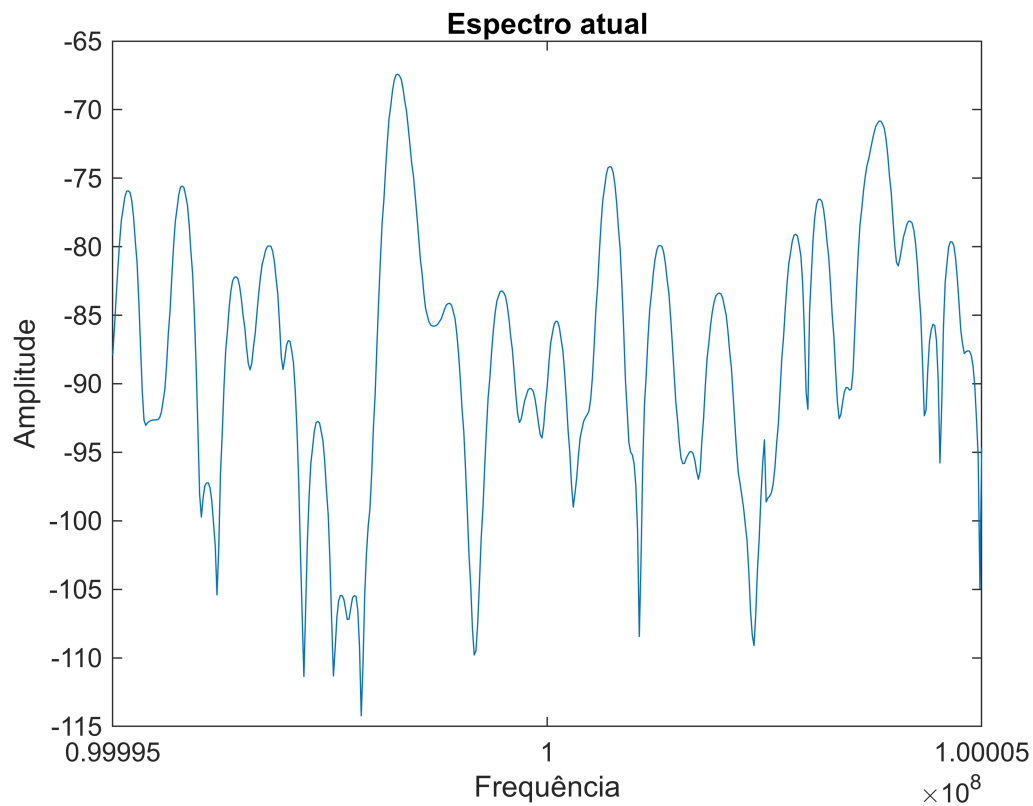
freq	value
9.9995e+07	-87.936
9.9995e+07	-85.869
9.9995e+07	-83.64
9.9995e+07	-81.498
9.9995e+07	-79.596

```

%% Plota o Trace
p = plot(trace.freq,trace.value,"DisplayName","value");

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

```



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

```
disp('Leitura do marcador:')
```

Leitura do marcador:

```
nivel = obj.getMarker(100002000, 1);
```

```
value =  
'-87.3036422729'
```

```
disp('Em 100.002 MHz o nível é:')
```

Em 100.002 MHz o nível é:

```
disp(nivel)
```

-87.3036422729

```
% Encerra ativamente a conexão  
obj.disconnect()
```

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

Pronto