Reverter a escravização inconsciente dos seres humanos pela tecnologia.

Tutorial: Introdução à Automação SCPI de Equipamentos de Teste com pyvisa

Publicado em 28 de março de 2021 por lui_gough

Quando se trata de realizar experimentos científicos, seus equipamentos de teste são seus *"olhos e ouvidos"*, medindo as grandezas que você está tentando observar. Muitas vezes, os compradores tendem a se concentrar em especificações como precisão e resolução, sem deixar de lado o preço. Todo o resto acaba sendo um pouco secundário.

No entanto, não é nenhum segredo que a maioria dos instrumentos de marcas renomadas de empresas de teste e medição possui algum grau de programabilidade. Os melhores instrumentos seguem uma série de padrões, possuem interfaces de programação remota, podem vir com drivers de software, exemplos de código e até mesmo um manual de programação com centenas *de* páginas.

No entanto, minha experiência mostra que **muitos bons instrumentos ficam parados na bancada** , enquanto os operadores ajustam os botões um a um, anotando as leituras em um caderno de laboratório, com as interfaces de controle remoto na parte traseira do instrumento desconectadas. Ou então, usam o software que acompanha o instrumento – alguns parecem ter sido desenvolvidos na época do Windows XP, mas quase sempre ouço reclamações de que "gostariam" de poder fazer algo um pouco diferente do que o software permite (por exemplo, usar um segundo ou terceiro instrumento, combinar instrumentos de outros fabricantes).

Quando vejo isso, suspiro *de* frustração – porque eu também já fui assim. Mas, em vez de desgastar os botões dos meus instrumentos, decidi pesquisar como automatizar esse processo e descobri que era **surpreendentemente fácil,** a ponto de me perguntar por que tão poucas pessoas se davam ao trabalho de tentar. Talvez seja o extenso manual de programação e a sintaxe SCPI, que não é comum em outros lugares.

Desde então, **não consigo** imaginar realizar a maioria dos experimentos que fiz para este blog sem usar a automação SCPI por meio *do pyvisa* . Tê-lo à disposição é como ter meu próprio assistente de laboratório que nunca se cansa, que nunca comete erros, que trabalha mesmo quando estou dormindo... é muito poderoso, mas também potencialmente perigoso.

Como resultado, decidi criar este tutorial que reúne toda a minha experiência em um só lugar para aqueles que desejam aproveitar ao máximo seus instrumentos compatíveis com SCPI. Pressupõe-se um conhecimento básico de Python 3 – não se preocupe, é bem fácil de aprender em comparação com a maioria das outras linguagens de programação.

Aviso: O tutorial a seguir é fornecido de boa fé. O autor não se responsabiliza por quaisquer erros, omissões ou danos, independentemente de como ocorram, resultantes do uso ou da impossibilidade de uso do conteúdo fornecido. A automação pode ser perigosa – certifique-se de ter considerado os riscos que podem surgir no pior cenário possível antes de iniciar o uso autônomo de automação para sequências de teste.

O que são VISA e *pyvisa?*

Quando se trata de automação de equipamentos de teste, você provavelmente já ouviu falar do termo " VISA ". *Não, não tem nada a ver com cartões de crédito* – na verdade, significa Virtual Instrument Software Architecture (Arquitetura de Software de Instrumento Virtual) e é uma API padrão da indústria usada para comunicação entre o computador e os instrumentos. Existem muitas **versões** diferentes de VISA – a mais comum é a NI-VISA da National Instruments, mas também existem as bibliotecas de E/S da Keysight, a TekVISA da Tektronix, a VISA da Rohde & Schwarz e outras. Embora devam ser interoperáveis até certo ponto, o software desenvolvido para uma determinada versão de VISA pode apresentar dificuldades de funcionamento com outra (na minha experiência), o que pode exigir a instalação de múltiplas versões de VISA, com uma designada como VISA primária. Ocasionalmente, isso pode resultar em um conflito que interrompe completamente a conectividade com o instrumento.

A função da VISA é servir de interface entre o seu programa aplicativo (que pode ser escrito em C ou qualquer outra linguagem para a qual existam bibliotecas disponíveis para a sua VISA) e o dispositivo, através do sistema operacional e seus drivers.

Seu programa comandaria o instrumento para executar determinadas tarefas – isso geralmente seria feito por meio de uma linguagem chamada SCPI (Standard Commands for Programmable Instruments), pronunciada “skippy”, *como o canguru-da-mata* . Esses comandos são baseados em strings ASCII e são enviados ao instrumento – mais detalhes sobre isso em uma seção posterior.

A automação poderia ser feita através da escrita de um programa compilado para a interface VISA e que utilizasse comandos SCPI, mas isso pode ser bastante limitante. Na maioria dos casos, as interfaces VISA comerciais operam em sistemas Windows de forma quase universal, o que significa que os programas não seriam compatíveis com outras plataformas. As linguagens disponíveis geralmente se limitariam a C ou suas variantes. O desenvolvimento seria bastante complexo, com o uso de comandos SCPI que podem ser específicos de cada instrumento, limitando a capacidade de portar o programa diretamente para outros instrumentos.

Como resultado, existem também outros métodos de automação – os drivers IVI-COM e IVI-C tentam abstrair a funcionalidade do instrumento para um nível superior, resolvendo problemas de intercambialidade entre diferentes modelos de instrumentos. Soluções mais sofisticadas, como o National Instruments LabVIEW, também fornecem uma interface gráfica de programação e são pioneiras no uso de drivers de instrumentos, que também abstraem a funcionalidade para um nível superior, permitindo que a substituição de instrumentos ocorra com mais facilidade.

Pessoalmente, nunca me entusiasmei muito com essas linguagens. Embora eu soubesse programar em C, a quantidade de trabalho necessária para configurar o ambiente de desenvolvimento, entender a documentação da biblioteca VISA e o fato de geralmente estar vinculada a uma distribuição VISA específica a tornavam pouco atraente. O fato de não ser multiplataforma também foi uma grande decepção. Por outro lado, o LabVIEW é um gasto injustificado e ocupa um espaço enorme em termos de tamanho de instalação.

Por isso, optei por automatizar usando *o pyvisa* . Ele funciona como uma "camada intermediária" entre o seu programa Python e uma camada VISA (NI-VISA ou *pyvisa-py*). O fato de permitir a programação em Python é uma grande vantagem, já que possui um bom suporte de bibliotecas, diversos recursos e é fácil de aprender. A capacidade de interagir com *o pyvisa-py* também significa que ele pode operar facilmente no Linux. Por fim, já mencionei que é **gratuito** e de código aberto?

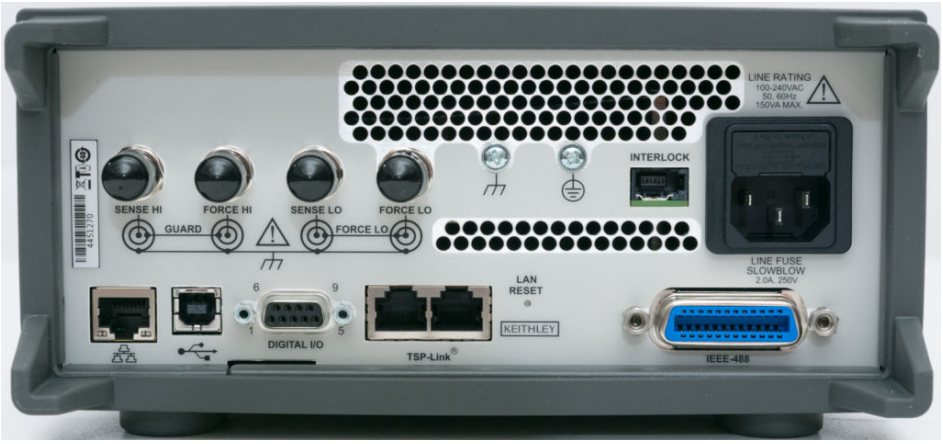
Os mais experientes me perguntaram: "Já que alguns instrumentos podem ser controlados por sockets TCP/IP ou serial RS-232, por que se preocupar com VISA?" Em alguns aspectos, essa é uma boa pergunta, mas acredito que haja um ótimo motivo para você ainda considerar o uso *do pyvisa* para esses instrumentos.

- Isso garante consistência quando se trabalha com vários instrumentos.
- Ele lida com timeouts, formata comandos e analisa respostas para você, para que você não precise converter uma string como “6.4892345E+05” em um número, por exemplo.
- Isso facilita a transição de uma interface para outra – o que acontece quando seu programa de socket LAN precisa, de repente, ser executado usando uma interface USB-TMC? Com *o pyvisa* , é tão fácil quanto alterar um Nome de Recurso VISA, mas usar sockets é um caminho completamente diferente.
- Isso também permite utilizar protocolos mais sofisticados em vez de apenas uma simples conexão de soquete. O uso do VXI-11 permite a transmissão de alguns sinais no estilo IEEE488, enquanto o HiSLIP permite a operação em pipeline, o que pode melhorar a velocidade. Melhor ainda, você não precisa se preocupar com a implementação desses protocolos, pois ela é gerenciada pela camada VISA.

Usar *o pyvisa* não resolve o problema de comandos SCPI serem, às vezes, específicos de cada dispositivo, mas isso raramente é um problema para experimentos pontuais ou em laboratórios domésticos, onde você pode não ter condições de adquirir novos instrumentos com frequência. Além disso, existem alguns comandos básicos que são comuns a todos os instrumentos – a maioria das operações pode ser feita com esses comandos.

Conexões da camada física do instrumento

A forma como os instrumentos se conectam aos hosts mudou ao longo dos anos; no entanto, existem alguns tipos de conexões na camada física, cada uma com suas próprias vantagens e desvantagens.

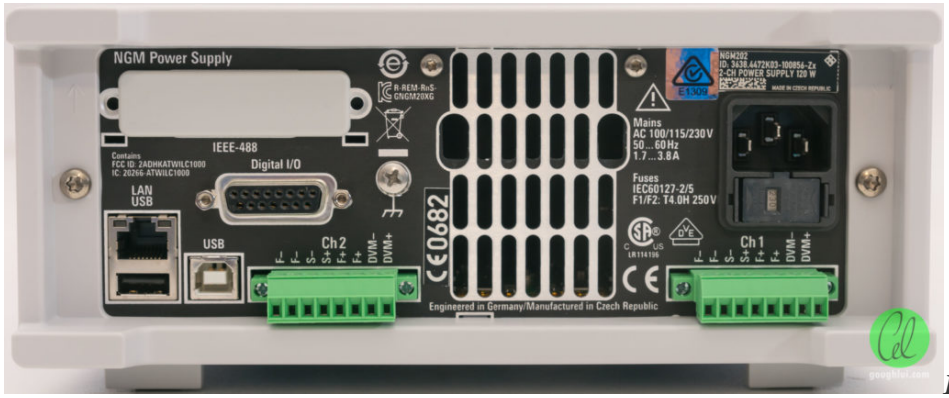


O mais antigo seria o General Purpose Interface Bus , ou GPIB, também conhecido como IEEE-488. Trata-se de um barramento paralelo com um conector micro-ribbon de 24 contatos, semelhante em estilo aos conectores Centronics usados anteriormente em impressoras com porta paralela, porém com fixação por parafuso e capacidade de empilhamento. Este barramento evoluiu do HP-IB e pode suportar até 15 dispositivos em um único segmento de 20 metros, e até 31 dispositivos no total com extensores ativos.

Este barramento específico é agora considerado uma forma de conexão "legada", que persiste principalmente para garantir a interoperabilidade de instrumentos com configurações de teste mais antigas. Em termos de velocidade de transferência de dados, não é tão rápido quanto as formas mais recentes de conectividade e tem um custo considerável, já que controladores e cabos podem facilmente custar mais da metade do preço de um instrumento! Ele possui uma **vantagem fundamental** : linhas dedicadas que podem ser usadas para garantir que vários instrumentos no barramento sejam acionados de forma síncrona para realizar medições simultaneamente.

A próxima opção seria a modesta porta serial RS-232, também conhecida como porta "COM" no mundo dos PCs IBM. Essa porta é comumente usada para equipamentos de terminal de dados (modems) e mouses, mas também pode ser usada com alguns instrumentos que aceitam comandos SCPI via serial. Essa porta está se tornando rara em alguns PCs modernos, mas adaptadores USB para serial podem ser usados para fazer a interface com esses dispositivos. As taxas de dados são relativamente limitadas, com taxas de transmissão comuns variando de 300 bps a 115.200 bps para a maioria dos dispositivos. A compatibilidade das configurações (em termos de taxa de transmissão, bits de dados, paridade e bits de parada) e da configuração do cabo (9 pinos/25 pinos, null-modem ou direto) é importante para a comunicação. A conectividade RS-232 é do tipo host para dispositivo, exigindo uma porta separada para cada dispositivo. Embora existam linhas especiais de controle de fluxo e status, elas geralmente não são usadas para fins de sincronização ou disparo; em vez disso, todos os comandos fluem como dados ASCII pela conexão serial.

Existem alguns instrumentos que se conectam via USB, mas são reconhecidos como uma porta serial usando a classe CDC (Communications Device Class). Nesses casos, você pode interagir com eles como se estivessem conectados por uma conexão serial, mas, alternativamente, alguns instrumentos podem ser configurados para operar via USB-TMC, o que pode ser preferível.



***Imagens** – os painéis traseiros de um SourceMeter Keithley 2450, uma carga eletrônica CC B&K Precision Modelo 8600 e uma fonte de alimentação Rohde & Schwarz NGM202. Você consegue ver todas as interfaces mencionadas?*

A partir daí, a maioria dos instrumentos optou pela conectividade USB, que é muito mais conveniente, pois se trata principalmente de uma operação "plug-and-play", sem necessidade de configuração ou drivers especiais (além de selecionar USB como modo de conectividade no menu do dispositivo). A conectividade USB foi padronizada por meio da USB Test and Measurement Class (ou USB-TMC, na sigla em inglês) e do USB488. Isso permite que os dispositivos funcionem com um driver padrão, emulando algumas das capacidades de sinalização de status do GPIB por meio de um formato de mensagem especial. O uso de USB também costuma significar latência mais consistente – o USB consulta os dispositivos a 1 kHz e, embora seja um barramento compartilhado,

geralmente possui largura de banda suficiente para o controle do instrumento. As limitações estão principalmente relacionadas ao barramento USB – por exemplo, geralmente se aplicam distâncias máximas de cinco metros entre o dispositivo e a porta, e uma cascata com no máximo cinco hubs de profundidade, embora existam algumas soluções que podem estender essas distâncias. O barramento também não é isolado, o que pode ser problemático em algumas aplicações, embora existam extensores, conversores de mídia e isoladores proprietários disponíveis. Infelizmente, com cabos de baixo custo e certificação relativamente frouxa de dispositivos interconectados, alguns dispositivos podem se tornar pouco confiáveis diante de forte ruído EMI ou incompatibilidades de chipset, especialmente em distâncias maiores.

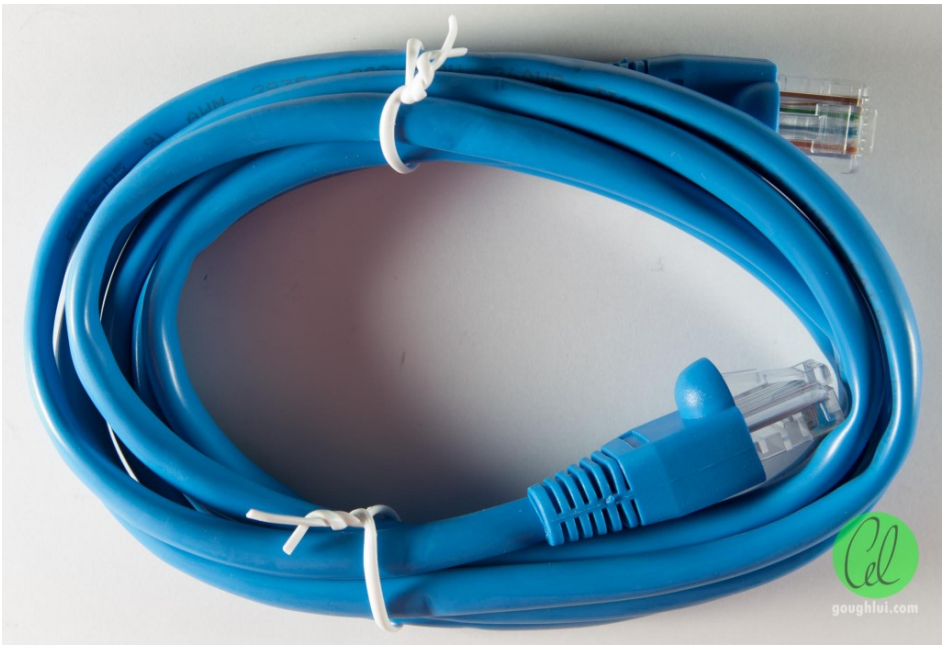
A forma mais moderna de conectividade para instrumentos é o Ethernet, tornando o instrumento praticamente uma "Internet das Coisas". Essa conectividade era considerada bastante complexa e cara, exigindo a implementação de uma camada física (PHY) Ethernet, camada de acesso MAC e pilha de rede TCP/IP completa para permitir o controle básico em nível de "soquete". Os benefícios, no entanto, são múltiplos, já que um dispositivo baseado em IP pode ser facilmente compartilhado com outros na mesma rede, pode ser acessado de forma semelhante a qualquer outro serviço baseado em IP (por exemplo, por meio de uma VPN) e pode compartilhar largura de banda com outros instrumentos e dispositivos no mesmo link de rede. Além disso, oferece a vantagem de utilizar conectividade Ethernet isolada por transformadores em ambas as extremidades e com alcance de até 100 metros entre o switch mais próximo e o dispositivo. Instrumentos de ponta com capacidade Ethernet são compatíveis com LXI (LAN Extensions for Instrumentation), que pode oferecer mais recursos, incluindo interface web, descoberta mDNS, protocolos VXI-11/Hi-SLIP e sincronização de tempo. Apesar disso, a conectividade Ethernet geralmente apresenta uma desvantagem significativa: a latência, que pode variar bastante dependendo da rede. Isso pode afetar a sincronização do acionamento dos dispositivos. Outro problema é que a maioria dos dispositivos não possui segurança robusta e, portanto, só pode ser usada em uma rede segura com nós confiáveis.

Vale ressaltar também que existem instrumentos que utilizam USB e Ethernet, mas podem não ser compatíveis com os padrões relevantes e podem usar seus próprios protocolos. Consultar a documentação fornecida pelo fabricante geralmente esclarece se o instrumento é compatível com SCPI. Existem também outros barramentos específicos de fornecedores para controle remoto que não são padrão da indústria (por exemplo, Keithley TSP-Link), os quais podem ser usados em alguns casos específicos, "intermediando" o tráfego através de um instrumento para enviar comandos ao restante do barramento. Da mesma forma, existe a possibilidade de usar adaptadores e gateways para converter GPIB ou USB em Ethernet e vice-versa.

Pré-requisitos de hardware e software

Para trabalhar com *o pyvisa* , você precisa de um ou mais instrumentos compatíveis com SCPI e alguma forma de conectá-los ao seu computador. Isso pode ser praticamente qualquer equipamento de teste de gama média a alta (por exemplo, um multímetro digital, uma fonte de alimentação, um analisador de espectro, etc.), desde que possua uma conexão GPIB, serial RS-232, USB (TMC) ou Ethernet (LXI) e suporte o conjunto de comandos SCPI (consulte o manual do instrumento em caso de dúvida).

Você também precisará do hardware compatível para conectá-lo ao seu computador, como um controlador GPIB e cabos GPIB, um cabo RS-232 para uma porta RS-232 ou um adaptador USB, um cabo USB A para B ou um cabo Ethernet. Naturalmente, o tipo de conexão e cabo deverá ser compatível, a menos que um adaptador ou gateway especial seja usado para fazer a conexão.



Antes de conectar o dispositivo, pode ser necessário configurá-lo. Certifique-se de selecionar a interface desejada no menu do dispositivo, pois a maioria dos aparelhos suporta apenas uma interface por vez.

- Para conexões GPIB, você precisará configurar um endereço (para dispositivos, geralmente de 1 a 30) com um único barramento, normalmente limitado a 20 metros de comprimento total do cabo e 15 dispositivos.
- No caso de conexões RS-232, observe a taxa de transmissão (baud rate), os bits de dados, a paridade e os bits de parada, embora geralmente a taxa de transmissão seja a mais importante, já que os parâmetros restantes costumam ser 8/N/1.
- Para conexões USB-TMC, o processo é praticamente "plug-and-play", desde que o software e os drivers estejam instalados corretamente.
- Por fim, para Ethernet, você precisará saber se deseja executar a autoconfiguração DHCP ou se irá alocar um endereço IP estático ao instrumento (e, em caso afirmativo, o endereço IP estático, a máscara de sub-rede, o gateway e os endereços DNS). A configuração incorreta desses endereços pode causar conflitos de rede ou impedir que o instrumento seja acessado pelo seu computador. Você pode optar por conectar seus instrumentos Ethernet à sua rede local (LAN) por meio de um switch, embora seja preciso ter cuidado para garantir que apenas dispositivos confiáveis estejam nessa rede, pois a maioria dos instrumentos Ethernet **não possui segurança** e, portanto, qualquer dispositivo na LAN pode potencialmente assumir o controle ou reconfigurar seu instrumento! Obviamente, os instrumentos não devem ser conectados diretamente à Internet!

A melhor coisa sobre *o pyvisa* é que ele funciona igualmente bem na maioria dos principais sistemas operacionais. Para executá-lo, você precisará ter o Python 3.x instalado. No Windows, você pode usar um pacote "tudo-em-um" com muitas bibliotecas comuns pré-instaladas, como o WinPython (que eu uso por ser portátil) ou o Anaconda . No Linux, o Python geralmente já está instalado ou disponível através do gerenciador de pacotes.

```
sudo apt-get install python3 python3-pip
```

Depois de instalar o PyVisa, você precisará instalá *-lo*. No Windows, isso é tão simples quanto executar o seguinte comando:

```
pip install pyvisa
```

Mas no Linux, provavelmente você deve executar este comando diretamente e, em seguida, novamente com o prefixo *sudo* para garantir que ele seja instalado para o seu usuário e *para o root* , já que seu usuário pode não ter permissões de acesso ao hardware (e se você não conceder esse privilégio ao seu usuário, terá que executar scripts como *root* , o que pode ser potencialmente perigoso).

Observação: Se o seu sistema Linux tiver o Python 2.x, versão obsoleta, como interpretador Python padrão, você precisará usar ***o pip3*** em vez do *pip* e ***o python3*** em vez do *python* para o restante deste tutorial!

Se lhe disserem que a sua versão do pip está desatualizada, pode atualizá-la com

```
python -m pip install --upgrade pip
```

Por fim, como *o pyvisa* é apenas um "wrapper" para uma camada VISA, você precisará instalar uma camada VISA compatível para que ele se comunique. Se você estiver usando o Windows, o ideal é usar o pacote NI-VISA da National Instruments . É um software robusto, mas, de modo geral, é a camada VISA com melhor suporte — uma versão dele pode já estar instalada no seu computador pelo software que acompanha o seu instrumento. É possível evitar o uso do NI-VISA no Windows, mas, nesse caso, você precisará desenvolver seus próprios drivers para cada instrumento e assiná-los manualmente, o que é bastante trabalhoso.

No Linux, entretanto, você também precisará instalar *o pyvisa-py* , que fornece uma camada VISA baseada em Python, bem como *o pyusb* (para instrumentos USB-TMC) e *o pyserial* (para instrumentos conectados por porta serial). Isso pode ser feito usando o seguinte comando (com ou sem *sudo*):

```
pip install pyvisa-py pyusb pyserial
```

A esta altura, você já deve ter uma instalação pronta para uso. Para testar, usuários do WinPython devem abrir um prompt de comando do WinPython e digitar ` *python* `. Uma mensagem com a versão do Python será exibida. Usuários de Linux devem abrir uma janela de terminal e digitar `*python3* ` , e a mensagem com a versão do Python também será exibida.

A partir daí, você deverá conseguir digitar

```
importar pyvisa
```

e deverá retornar ao prompt, exatamente como no exemplo abaixo. Se isso acontecer, parabéns, sua instalação foi bem-sucedida e você pode digitar *exit()* para sair.

```
Python 3.6.6 (v3.6.6:4cf1f54eb7, 27 de junho de 2018, 03:37:03) [MSC v.1900
64 bits (AMD64)] no win32
Digite "ajuda", "direitos autorais", "créditos" ou "licença" para obter mais informações.
Informação.
>>> importar pyvisa
>>>
```

Observação: *Exemplos de código mais antigos podem usar a linha "import visa" em vez de "import pyvisa". Isso ainda é aceito, mas pode causar conflitos, pois outro pacote também usa o nome "visa". Portanto, recomenda-se usar "import pyvisa" daqui para frente. Para códigos mais antigos que dependem do nome "visa", você pode usar "import pyvisa as visa" como solução alternativa. Isso é indicado pelo erro que você pode receber em versões mais recentes do pyvisa.*

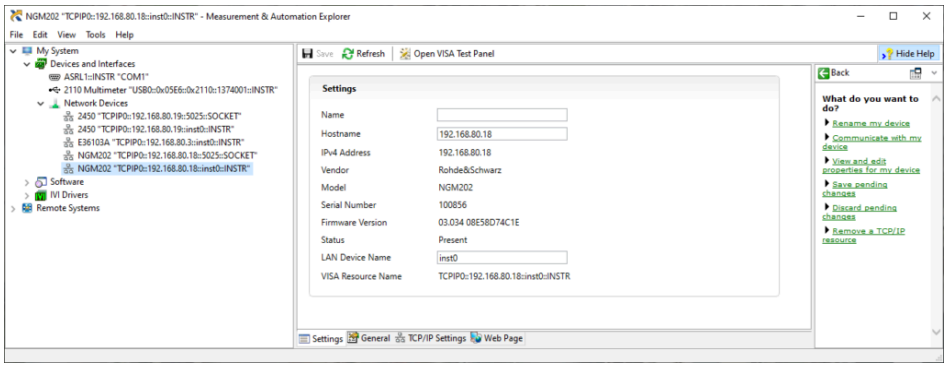
```
/usr/local/lib/python3.7/dist-packages/visa.py:23: FutureWarning:
O módulo de vistos fornecido pelo PyVISA está sendo descontinuado. Você pode
Substitua `import visa` por `import pyvisa as visa` para obter o resultado desejado.
mesmo efeito.

O motivo da descontinuação é o possível conflito com o
pacote de visto fornecido pela
https://github.com/visa-sdk/visa-python o que pode resultar em dificuldades
para depurar situações.
Aviso Futuro,
```

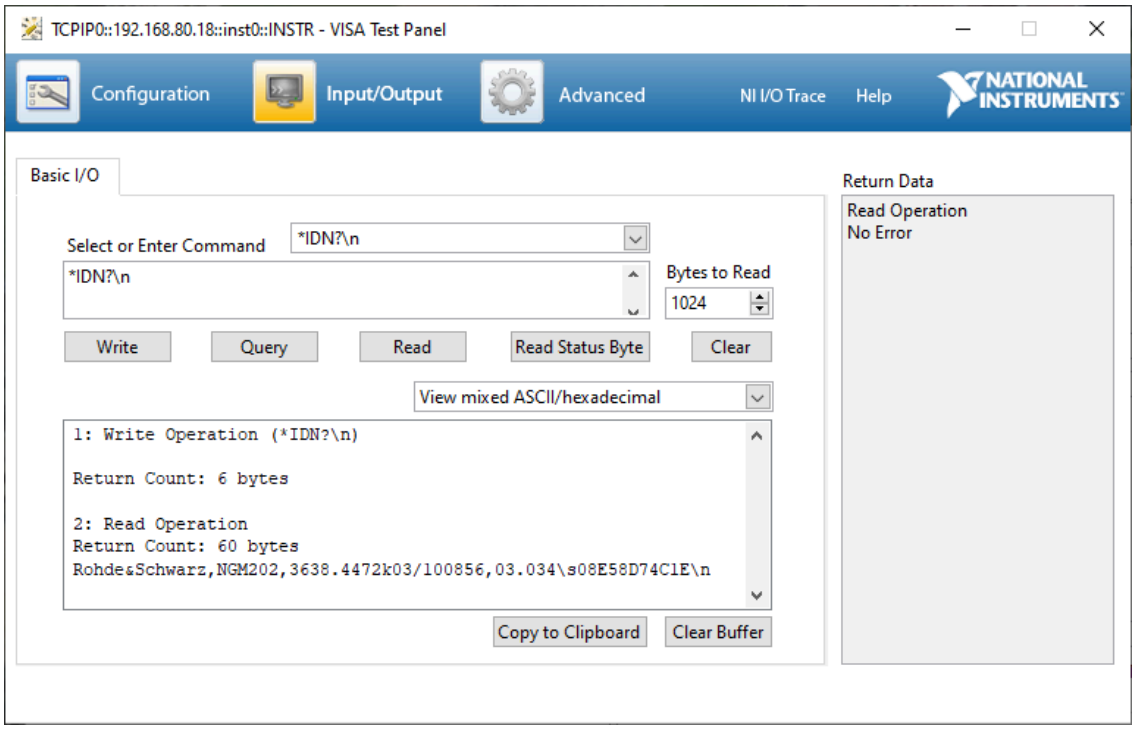
Nomes de recursos VISA

Para usar um dispositivo, primeiro você precisa saber como endereçá-lo. Cada dispositivo que pode ser acessado por meio da VISA possui um endereço exclusivo, conhecido como Nome de Recurso VISA. Existem algumas maneiras de verificar seus dispositivos e seus respectivos nomes de recurso.

Se você usa o Windows e o NI-VISA, então você já deve ter instalado uma ferramenta chamada NI-MAX. Essa ferramenta permite verificar os dispositivos e interfaces disponíveis expandindo a árvore no lado esquerdo. Clicar em um instrumento exibirá o nome completo do recurso.



Essa ferramenta específica também é bastante útil para verificar se seus instrumentos estão funcionando corretamente – clicar em *Abrir Painel de Teste VISA* e, em seguida, no botão *Entrada/Saída* na barra de ferramentas superior permite enviar/receber comandos diretamente para verificar o funcionamento do instrumento por meio do NI-VISA.



Para quem usa Linux ou não quer invocar o NI-MAX, também é possível verificar quais dispositivos estão conectados à sua máquina executando alguns comandos Python, como segue:

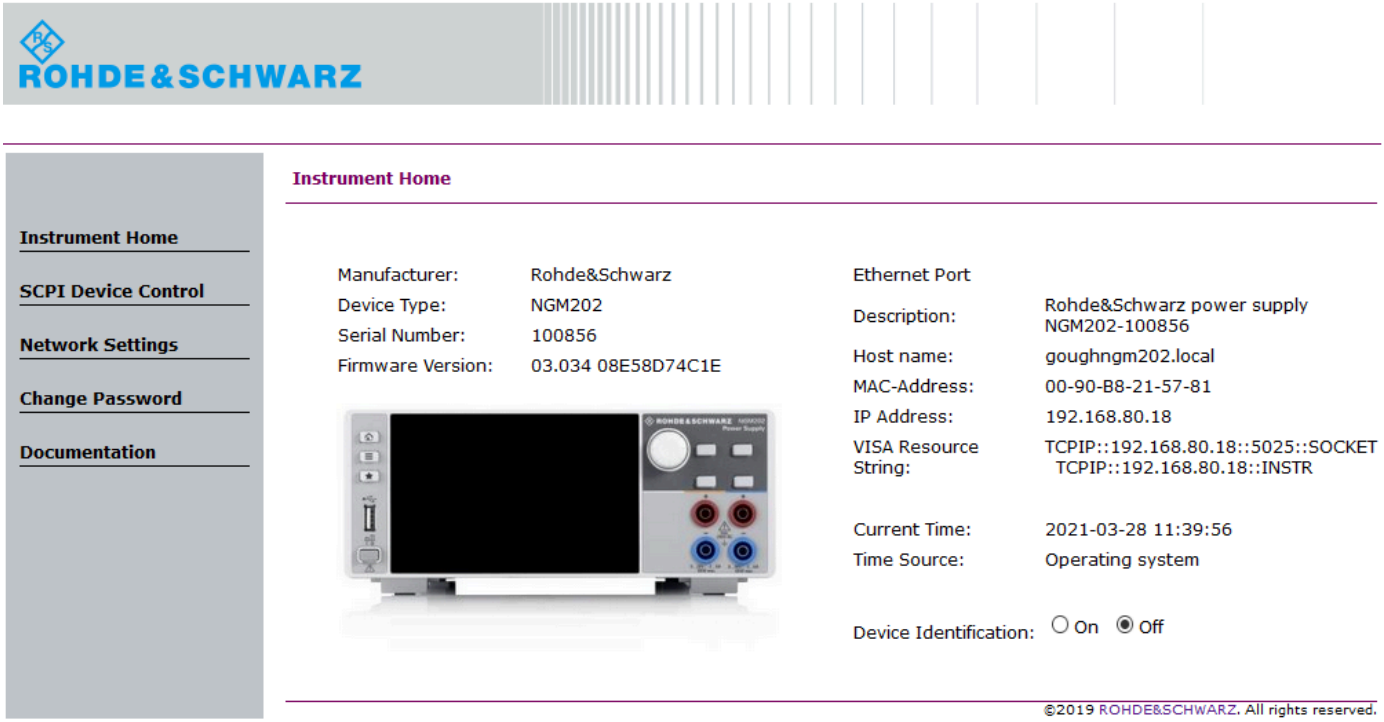
```
>>> importar pyvisa
>>> rm = pyvisa.ResourceManager()
>>> rm.list_resources()
('USB0::0x05E6::0x2110::1374001::INSTR',
'TCPIP0::192.168.80.18::inst0::INSTR',
'TCPIP0::192.168.80.19::inst0::INSTR',
'TCPIP0::192.168.80.3::inst0::INSTR', 'ASRL1::INSTR')
```

O resultado é um exemplo de uma máquina Windows executando NI-VISA. Executando o mesmo em um Raspberry Pi com Linux e *pyvisa-py* , as strings são ligeiramente diferentes –

```
>>> importar pyvisa
>>> rm = pyvisa.ResourceManager()
>>> rm.list_resources()
('USB0::65535::34816::602197010716810015::0::INSTR',)
```

Observe como a string de recurso USB usa notação decimal no Linux por meio *do pyvisa-py* e hexadecimal com o NI-VISA. Independentemente disso, essas strings geralmente são intercambiáveis. **Deve-se notar que as strings podem diferenciar maiúsculas de minúsculas** – o “insto” precisa estar em minúsculas, caso contrário, um erro pode ser lançado!

Por fim, para instrumentos LXI baseados em LAN, pode ser possível encontrar esses detalhes e algumas funcionalidades básicas de comando SCPI através de sua interface web, acessando o endereço IP diretamente em um navegador.



Alguns instrumentos não serão detectados automaticamente, especialmente alguns instrumentos GPIB, seriais e LAN (soquete). Isso significa que você precisará escrever seu próprio nome de recurso que corresponda ao instrumento específico, por exemplo:

```
'ASRL1::INSTR' para a porta serial 1

'GPIB::5' para o endereço de instrumento GPIB 5 (considerando um controlador)

'GPIB1::10' para o endereço de instrumento GPIB 10 (no primeiro controlador)

'TCPIP0::192.168.80.61::5025::SOCKET' para uma conexão de soquete TCP/IP
..................... para 192.168.80.61 na porta 5025 (SCPI-RAW)
```

... e a camada VISA tentará estabelecer uma conexão conforme as instruções, apresentando um erro caso a comunicação não possa ser estabelecida. Algumas dessas conexões específicas precisarão de configurações de parâmetros adicionais para funcionarem corretamente — consulte a seção Dicas e Truques no final para obter mais detalhes.

Tipos e formato de comandos SCPI

Talvez você tenha consultado o manual de programação do seu instrumento e se sentido perdido com a quantidade de informações – **não se preocupe** , muitas delas dizem respeito a detalhes específicos com os quais você só precisa se preocupar se usar um determinado comando.

Começando pelo básico: todos os comandos SCPI são sequências ASCII que você envia para o instrumento. Existem dois tipos de comandos SCPI: comandos de **configuração** e comandos **de consulta** . Os comandos de configuração instruem o instrumento a executar uma ação, enquanto os comandos de consulta solicitam que o instrumento responda com alguma informação. Alguns comandos estarão disponíveis apenas como comandos de configuração, outros apenas como comandos de consulta, enquanto que, frequentemente, os comandos podem estar disponíveis tanto como comandos de configuração quanto como comandos de consulta, permitindo que você

consulte qual valor foi definido anteriormente. **Um comando de consulta pode ser identificado pela presença do símbolo *"?"* .**

Programaticamente, um comando de configuração (set) envolve escrever uma string no instrumento, incluindo um caractere de terminação (geralmente *"\n"*), enquanto um comando de consulta (query) envolve escrever a string no instrumento com o caractere de terminação e, em seguida, **ler** uma resposta do instrumento (geralmente até o caractere de terminação ou a terminação é sinalizada por outros meios). Os instrumentos responderão a consultas válidas com os dados, embora algumas operações possam demorar mais do que outras. Consultas inválidas geralmente resultarão em um erro registrado, um bipe emitido e nenhuma resposta. **As consultas de leitura do instrumento têm um tempo limite padrão de 2000 ms** – quaisquer consultas que demorem mais para retornar resultarão em um erro de tempo limite, a menos que o tempo limite seja ajustado para um valor maior (consulte Dicas e Truques para obter mais informações). No entanto, o envio de comandos de consulta incorretos resultará em um tempo limite que pode exigir que você tome medidas manuais para se recuperar.

A maioria dos dispositivos só consegue processar comandos SCPI **em sequência** e um de cada vez. Portanto, é crucial que, ao emitir comandos de consulta, o resultado inserido no buffer de saída seja lido imediatamente após cada comando de escrita. Caso contrário, você poderá perder a sincronia com o instrumento e ler uma resposta incorreta! Além disso, há o **tempo** necessário para o computador, a camada VISA, o sistema operacional e a camada física transmitirem o comando ao instrumento, bem como o tempo que o instrumento leva para receber, processar e responder ao comando. Essa latência limita a velocidade de controle do instrumento.

Os comandos SCPI estão organizados em uma hierarquia que começa na raiz, representada pelo símbolo de dois pontos *(:)* . O primeiro nível abaixo da raiz é frequentemente conhecido como um "subsistema", o nível abaixo deste é geralmente conhecido como um comando ou subcomando, e todos são separados por dois pontos. Por padrão, os comandos são interpretados a partir do nível raiz.

Os comandos são listados em um manual de programação de forma semelhante à seguinte:

```
[:FONTE:]VOLTagem <Valor>
```

O próprio comando SCPI geralmente não diferencia maiúsculas de minúsculas. Os comandos podem ser abreviados omitindo todas as letras minúsculas. Os comandos podem ser ainda mais abreviados omitindo as seções entre colchetes que são "opcionais" – essas alternativas geralmente estão presentes para garantir uma conformidade estrita com os princípios da hierarquia SCPI e para melhorar a compatibilidade com o código escrito para outros instrumentos. Os comandos podem começar com um *":"* para garantir que sejam interpretados como provenientes da raiz; no entanto, isso nem sempre é necessário se você não usar concatenação de comandos *(;)* . Ao usar concatenação de comandos, os comandos subsequentes são tratados como se estivessem no mesmo nível. Muitos comandos aceitam um valor de entrada – no caso do comando acima, será a tensão, mas também poderia ser uma string. Os valores entre colchetes podem ser omitidos.

Consequentemente, os seguintes comandos são todos equivalentes (juntamente com suas versões em minúsculas):

```
FONTE: TENSÃO 10
FONTE: VOLT 10
FONTE:VOLTAGEM 10
SOUR:VOLT 10
VOLT 10
```

Alguns comandos são comuns e você os encontrará em praticamente todos os instrumentos do mesmo tipo (por exemplo, *MEAS:VOLT?*), enquanto outros podem ser específicos de um determinado modelo de instrumento. Às vezes, o manual de programação contém notas ou sugestões sobre isso e sobre os comandos preferenciais a serem usados (já que alguns comandos podem ser legados e não exploram todos os recursos dos instrumentos mais recentes).

Alguns comandos comuns e comandos de barramento assumem uma forma diferente, começando com um asterisco (*). Isso inclui comandos como *RST, *IDN, *STB, *OPC e outros. Esses comandos são comandos comuns do padrão IEEE 488.2 que geralmente funcionam na maioria dos instrumentos compatíveis com SCPI e executam funções definidas.

Depois de pegar o jeito, basta consultar o manual de programação para encontrar os comandos necessários para implementar os testes que você deseja realizar. Normalmente, isso envolverá apenas alguns comandos dentre a infinidade de opções disponíveis.

Escrevendo alguns programas simples *em pyvisa*

Esta seção apresentará alguns programas muito simples, cujo objetivo principal é fornecer uma referência inicial, ser um pouco instrutiva e dar-lhe alguma confiança para que você possa escrever seu próprio código.

Olá, mundo!

O primeiro programa que você geralmente aprende a escrever é um programa "Olá, Mundo!". Nesse caso, o equivalente para instrumentos musicais seria um programa que se conecta a um instrumento e recupera sua identidade.

```
importar pyvisa
gerenciador_de_recursos = pyvisa.GerenciadorDeRecursos()
Você pode alterar o nome da variável e o nome do recurso.
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::inst0::INSTR")
print(ins_ngm202.query("*IDN?"))
ins_ngm202.close()
```

Este programa simples começa importando *o pyvisa* e, em seguida, conectando-se ao gerenciador de recursos. A partir daí, conectamo-nos ao recurso com o nome VISA. Imprimimos o resultado de uma consulta **IDN?* no console antes de fechar e encerrar o programa.

A maioria das linhas deste programa serão comuns a quase todos os programas.

Exibir texto e emitir sinal sonoro

Talvez a sua ideia de "Olá Mundo" seja exibir algum texto – alguns instrumentos podem fazer isso com o comando *DISP:TEXT* . Da mesma forma, podemos fazer o instrumento emitir um bipe com o comando *BEEP* .

```
importar pyvisa
gerenciador_de_recursos = pyvisa.GerenciadorDeRecursos()
Você pode alterar o nome da variável e o nome do recurso.
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::inst0::INSTR")
ins_ngm202.write("DISP:TEXT \"Olá Mundo\"")
```

```
ins_ngm202.write("BIP")
ins_ngm202.close()
```

Controlar uma saída, medir tensão e corrente.

Começando com algo mais simples, que tal um exemplo que defina uma tensão e uma corrente, ligue a saída, espere cinco segundos, meça a tensão e a corrente e, em seguida, desligue a saída?

```
importar pyvisa
tempo de importação
gerenciador_de_recursos = pyvisa.GerenciadorDeRecursos()
Você pode alterar o nome da variável e o nome do recurso.
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::inst0::INSTR")
ins_ngm202.write("FONTE:VOLT 10")
ins_ngm202.write("SOUR:CURR 1")
ins_ngm202.write("OUTP 1")
tempo.dormir(5)
voltage = ins_ngm202.query_ascii_values("MEAS:VOLT?")
atual = ins_ngm202.query_ascii_values("MEAS:CURR?")
ins_ngm202.write("OUTP 0")
print("A tensão era "+str(voltage[0])+". A corrente era "+str(current[0])+".")
ins_ngm202.close()
```

Este programa é um pouco mais complexo, mas de fato realiza algo prático. Você notará que também importamos o módulo *de tempo, pois ele é útil para gerar atrasos. O subsistema SOURCE* do instrumento está sendo usado para definir a tensão e a corrente de saída, e o subsistema *MEASURE* está sendo usado para ler os valores, enquanto o subsistema *OUTPUT* controla o estado da saída.

Você notará que os comandos **`set`** são escritos usando a função `*.write*`, enquanto **as consultas** são escritas usando a função `*.query_ascii_values*` . Na verdade, existem várias maneiras de escrever funções de consulta – você pode usar um `*.write*` seguido de um `*.read*`, que retornará um objeto Python contendo a resposta bruta. No entanto, muitas consultas a instrumentos resultam em um ou mais valores numéricos separados por vírgulas, então, nesse caso, usar `*.query_ascii_values*` converterá o valor em uma lista Python de objetos numéricos – assim, em vez de trabalhar com uma string que pode dizer "2.9777E+02", você trabalha com o número real, permitindo que comparações e operações matemáticas sejam executadas diretamente no resultado. Mesmo que o valor retornado seja apenas um único número, a função retorna uma lista, então a notação de índice (ou seja, `*[0]*`) é usada para acessar o primeiro elemento da lista para impressão, enquanto a função `*str*` é usada para converter esse número de volta em uma string imprimível, já que o operador ` + ` espera tipos de dados correspondentes.

Embora este programa simples possa funcionar, ele apresenta algumas deficiências que serão explicadas no próximo exemplo, um pouco mais complexo.

Varredura IV

Vamos imaginar que você queira realizar uma varredura de um LED usando uma fonte de alimentação de 0 a 3V em incrementos de 1mV com um limite de corrente de 20mA e registrar os dados obtidos juntamente com o tempo em segundos Unix. Como isso poderia ser feito?

Bem, não é tão difícil assim, mas há algumas coisas que você provavelmente deveria fazer que talvez não sejam tão óbvias – aqui está a minha solução:

```
importar pyvisa
importar csv
tempo de importação
gerenciador_de_recursos = pyvisa.GerenciadorDeRecursos()
Você pode alterar o nome da variável e o nome do recurso.
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::inst0::INSTR")

ins_ngm202.write("*RST")
ins_ngm202.write("SYST:REM")
ins_ngm202.write("INST:NSEL 1")
ins_ngm202.write("FONTE:VOLT 0")
ins_ngm202.write("SOUR:CURR 0.02")
ins_ngm202.write("OUTP 1")
ins_ngm202.query("*OPC?")

com open("log.csv","w",newline="") as csvfile :
  logfile = csv.writer(csvfile)
  logfile.writerow(["Tempo","Tensão Solicitada","Tensão Lida","Atual Lida"])
  para x no intervalo (0,3001,1):
    ins_ngm202.write("SOUR:VOLT "+str(x/1000.0))
    ins_ngm202.query("*OPC?")
    readback = ins_ngm202.query_ascii_values("READ?")
    logfile.writerow([time.time(),(x/1000.0),readback[0],readback[1]])

ins_ngm202.write("OUTP 0")
ins_ngm202.write("SYST:LOC")
ins_ngm202.close()
```

Este código amplia significativamente o exemplo acima – agora também importei o módulo *csv* para permitir a gravação de um arquivo de log .csv (embora isso também pudesse ser feito usando um descritor de arquivo comum).

Um dos problemas com o código anterior é que **não sabemos qual era o estado anterior do instrumento** . Ele poderia estar em praticamente qualquer configuração, então definir apenas algumas opções não garantiria que a saída seria a esperada. Consequentemente, é uma **boa prática** emitir um comando **RST* para redefinir o instrumento para os valores padrão listados no manual, de modo que tudo esteja em um estado conhecido.

Outro problema é que o código anterior foi executado, mas não tratou do estado do instrumento – alguém pode ter mexido no painel frontal, alterando as configurações durante a execução, o que resultou em comportamentos inesperados. Existe um comando (*SYST:REM*) que coloca a unidade em modo de controle remoto, bloqueando o painel frontal até que uma chamada *SYST:LOC* correspondente seja emitida ou que ele seja desbloqueado pelo painel frontal. Nem todos os instrumentos implementam esse comando.

Para instrumentos multicanal, cada canal é considerado um “instrumento” independente e um comando *INST:NSEL* ou *INST:SEL* é necessário para selecionar a qual instrumento os comandos seguintes serão direcionados. Neste caso, o comando indica que os comandos a seguir terão efeito no Canal 1; no entanto, alguns comandos são “globais” e afetarão todos os canais (por exemplo, *OUTP:GEN 0,* que desativa todas as saídas), portanto, deve-se ter cuidado ao executar comandos!

Por fim, é importante observar que o envio de comandos separados *MEAS:VOLT?* e *MEAS:CURR?* pode não produzir o resultado esperado por alguns motivos: um deles é que pode haver um atraso suficiente entre as duas chamadas, impedindo a obtenção de uma leitura síncrona de tensão e corrente. Outro motivo é que alguns instrumentos podem não **medir** no momento da instrução, retornando o valor da corrente. Assim, se o instrumento processar os comandos rapidamente, um loop pode ler o mesmo valor várias vezes.

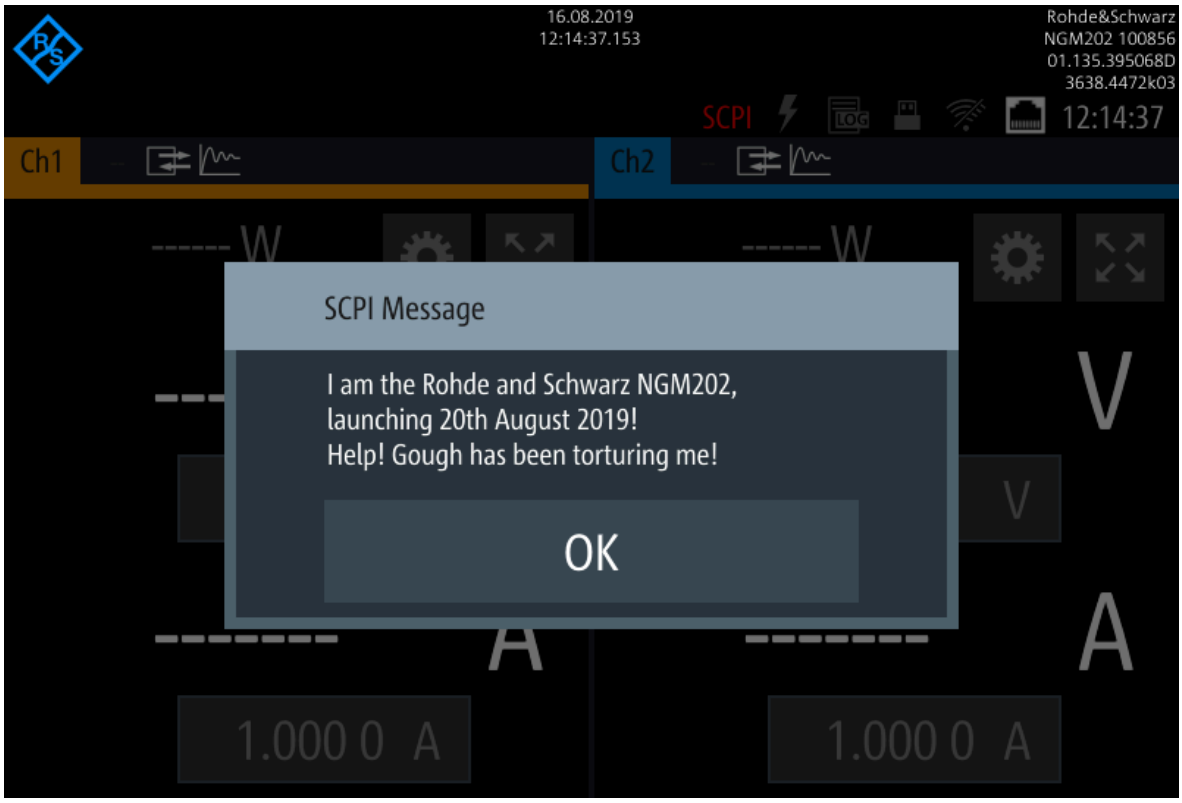
É aqui que a leitura do manual do seu instrumento se torna crucial – alguns instrumentos utilizam *FETC?* ou *READ?* para ler medições existentes, enquanto *MEAS* sempre gera uma nova medição. Outros podem precisar ser configurados para disparo por software usando *TRIG:SOUR BUS* , seguido pelos comandos *INIT* e **TRG* para disparar uma medição e, em seguida, um *FETC?* para lê-la. No caso do NGM202 com o firmware mais recente, *READ?* é uma chamada bloqueante que retorna um novo conjunto de valores a cada chamada, incluindo tensão e corrente, tornando-a mais rápida do que usar chamadas de medição separadas.

Faça uma captura de tela

Os instrumentos não se limitam a retornar apenas valores ASCII. Na verdade, muitos instrumentos também são capazes de retornar dados binários – um tipo comum de dado binário são as imagens de captura de tela. Este programa é um exemplo de programa capaz de capturar uma imagem da tela e gravá-la em um arquivo.

```
importar pyvisa
gerenciador_de_recursos = pyvisa.GerenciadorDeRecursos()
Você pode alterar o nome da variável e o nome do recurso.
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::inst0::INSTR")
f = open("screenshot.png","wb")
ins_ngm202.write("HCOP:DADOS?")
dados=ins_ngm202.ler_bruto()
# Os dados retornados são prefixados com #nxxxxx, onde n é o número de bytes em
# Campo de comprimento, xxxxxx são os bytes de dados seguidos por \n. Índice
A notação # é usada para remover o cabeçalho e o rodapé.
f.write(data[(data[1]-46):-1])
f.close()
ins_ngm202.close()
```

Este programa difere ligeiramente dos demais: ele **escreve** um comando de consulta e, em seguida, chama a função `*read_raw*` para recuperar os dados da imagem do subsistema *HARDCOPY* . Esses dados são codificados de uma maneira específica: `*#nxxxxx*`, onde `n` é um dígito ASCII que indica o número de bytes no campo de comprimento e `x` são os dígitos ASCII que representam o número de bytes. O registro inteiro é seguido por um caractere de terminação. A indexação em Python é usada para "fatiar" os dados retornados, removendo o cabeçalho e o caractere de terminação, e gravando os dados binários em um arquivo chamado `*screenshot.png*` (o que seria válido se o seu instrumento retornasse dados PNG; alguns também retornam BMP).



A partir desses exemplos, deve ficar claro que fazer algo funcionar com *o pyvisa* é relativamente fácil, mas torná-lo **à prova de falhas** requer um pouco de cuidado e compreensão de como o instrumento funciona. A próxima seção apresentará algumas dicas e truques adicionais que não abordei nestes exemplos. Lembre-se de que esta é apenas uma **introdução** , portanto, não abordarei as complexidades (como registros com status questionável), que posso reservar para um tutorial mais avançado.

Você também encontrará, espalhados por muitos dos meus blogs, alguns códigos *em pyvisa* , então espero que isso ajude a entender melhor as listagens desses programas. Vale ressaltar que nem sempre sigo as melhores práticas, pois às vezes é possível usar alguns atalhos (principalmente para experimentos pontuais), desde que você conheça bem seu instrumento e sua configuração.

Dicas e truques gerais de automação SCPI

Esta seção reúne algumas dicas e truques para trabalhar com instrumentos. Algumas delas são consideradas boas práticas, enquanto outras são sugestões que compilei ao trabalhar com diversos modelos de instrumentos de diferentes fabricantes.

Definindo valores

Embora os exemplos acima mostrem uma configuração de valores bastante simples, é importante observar que você está assumindo **implicitamente** que o instrumento executou o comando. Se, por algum motivo, o instrumento não conseguir executar o comando (por exemplo, erro de comunicação, estouro ou corrupção de buffer), seu programa não saberá disso, a menos que execute alguma operação adicional. Uma maneira de fazer isso seria verificar a fila de erros (veja a subseção posterior).

Mas outro método muito simples seria consultar o valor definido logo em seguida para verificar se ele corresponde (com uma pequena margem, devido a possíveis erros induzidos por ponto flutuante). Por exemplo, se você executar um *SOUR:VOLT 10* , poderia prosseguir com um *SOUR:VOLT?* e, com sorte, a resposta será

1.00000+E01 ou algo semelhante. Isso é considerado uma boa prática, porém, eu o considero desnecessário com instrumentos confiáveis.

Você também pode aproveitar a capacidade de automação para definir valores em ordem aleatória – por exemplo, quando uma varredura em uma direção pode produzir resultados tendenciosos, você pode varrer na direção oposta ou em ordem aleatória. Você também pode aproveitar a oportunidade para fazer várias leituras e calcular a média para obter um resultado mais confiável, além de ter as estatísticas correspondentes.

Sincronização, tempos limite e atrasos

Outro problema potencial é que seu programa não sabe exatamente quando um determinado comando "set" é executado, já que você pode estar enviando uma sequência de comandos "set" um após o outro.

Para isso, existe a consulta **OPC?,* que sempre retornará *1* quando as operações até aquele ponto forem concluídas. Alternativamente, **WAI* também é um comando que pode ser usado de forma semelhante, embora eu prefira o primeiro.

Outro problema que você pode encontrar é a lentidão dos instrumentos ou sequências de comandos longas, que podem fazer com que seu instrumento demore mais do que o esperado para responder. Isso resultará em um erro de tempo limite excedido se a resposta demorar mais do que o tempo limite padrão de 2000 ms.

Você pode reconfigurar o tempo limite usando a diretiva *.timeout* , por exemplo.

```
ins_ngm202.timeout = 10000
```

reconfiguraria o tempo limite para *ins_ngm202* para dez segundos.

Existe também a possibilidade de seu instrumento lento sofrer um estouro de buffer se você enviar muitos comandos de configuração em sequência sem incluir um **OPC?* na sequência. Geralmente, é considerado uma má prática adicionar atrasos fixos (por exemplo, usando *time.sleep*), mas às vezes pode ser a solução mais fácil se o desempenho não for crítico.

Fila de erros

Quando ocorrem erros durante a execução de comandos SCPI, por exemplo, devido a um comando digitado incorretamente ou a um erro de comunicação, a maioria dos dispositivos emitirá um bipe e poderá exibir um indicador de erro na tela.

Você pode consultar a fila de erros para descobrir a causa do erro usando a consulta *SYST:ERR?,* que retornará o erro mais recente e o removerá da fila. Para recuperar toda a fila de erros, é necessário executar o comando novamente até obter o retorno "0" (sem erros) para o primeiro elemento. A utilidade das mensagens pode variar de instrumento para instrumento, mas é uma boa prática verificar a fila de erros em seu programa para garantir que seu instrumento esteja funcionando corretamente.

Vale ressaltar também que alguns erros não aparecem necessariamente na fila de erros e podem, em vez disso, definir determinados bytes de status. Consulte o manual para obter detalhes, pois cada instrumento usa seus bytes de status mapeados em bits de maneira diferente.

Linguagem e Sintaxe

Alguns dispositivos são capazes de operar tanto em SCPI quanto em outro idioma. O exemplo mais comum são os instrumentos Keithley com TSP integrado. Para colocá-los no modo SCPI, utilize o comando **LANG SCPI* para alternar o conjunto de comandos remotamente. Como alternativa, você pode fazer a alternância pelo painel frontal ou pela interface web.

Existem também alguns instrumentos (como o Tektronix PA1000) que **exigem explicitamente que** cada comando comece com dois pontos, caso contrário, será ignorado.

A maioria dos instrumentos é um tanto exigente quanto ao caractere de terminação, que geralmente é um caractere de nova linha ($\backslash n$). O envio de comandos com um $\backslash r \backslash n$ pode resultar em erros.

Vários instrumentos

Possui vários instrumentos de diferentes fornecedores? Sem problemas, basta declarar cada um deles com seu próprio nome e a respectiva string de recurso, assim:

```
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::INST0::INSTR")
ins_k2110 = resource_manager.open_resource("USB0::0x05E6::0x2110::1374001::INSTR")
ins_bk8600 = resource_manager.open_resource("USB0::0xFFFF::0x8800::602197010716810015::INSTR")
```

Em seguida, você pode escrever para cada um usando o nome correspondente –

```
print("Configuração - NGM202")
ins_ngm202.write("OUTP:GEN 0")
ins_ngm202.write("INST:NSEL 1")
ins_ngm202.write("SENS:VOLT:RANG:AUTO 0")
ins_ngm202.write("SENS:VOLT:RANG 5")
ins_ngm202.write("SENS:CURR:RANG:AUTO 0")
ins_ngm202.write("SENS:CURR:RANG 10")
ins_ngm202.write("OUTP:MODO SOUR")
ins_ngm202.write("FONTE:VOLT 5.000")
ins_ngm202.write("SOUR:CURR 6.000")
ins_ngm202.write("INST:NSEL 2")
ins_ngm202.write("VOLT:DVM 1")
ins_ngm202.write("FONTE:VOLT 0.000")
ins_ngm202.write("SOUR:CURR 3.000")
ins_ngm202.write("OUTP 1")
ins_ngm202.query("*OPC?")

print("Configuração - K2110")
ins_k2110.write("FUNC \"CURR:DC\"')
ins_k2110.write("CURR:NPLC 10")
ins_k2110.write("TRIG:SOUR BUS")
ins_k2110.write("CURR:DC:RANG 10")
ins_k2110.query("*OPC?")

print("Configuração - BK8600")
ins_bk8600.write("SOUR:FUNC CURR")
ins_bk8600.write("CURR 0")
ins_bk8600.write("REM:SENS 1")
ins_bk8600.write("INP 1")
ins_bk8600.query("*OPC?")
```

No exemplo acima, meu script configura um experimento que realizei para o RoadTest do MAX77751, no qual precisei que minha fonte de alimentação Rohde & Schwarz NGM202, a carga eletrônica CC B&K Modelo 8600 e o multímetro digital Keithley 2110 funcionassem em conjunto. A grande vantagem do SCPI é que todos esses

componentes podem ser de fornecedores diferentes e, ainda assim, funcionar juntos de forma praticamente perfeita.

Observe também que, no exemplo acima, onde os argumentos da sua função precisam do caractere de aspas duplas e você está usando strings entre aspas duplas, você precisará escapá-lo com um caractere de barra invertida “\” .

Conexões HiSLIP

Alguns instrumentos de alto desempenho oferecem um protocolo Ethernet com capacidade de pipeline chamado HiSLIP, abreviação de High Speed LAN Instrument Protocol (Protocolo de Instrumento LAN de Alta Velocidade). Para dispositivos que suportam HiSLIP, uma varredura de portas mostraria a porta 4880 aberta.

Como resultado, os dispositivos Ethernet podem ser conectados de diversas maneiras diferentes –

```
# Conecte usando um soquete bruto
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::5025::SOCKET")
# Conecte-se usando o VXI-11 padrão
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::INST0::INSTR")
# Conecte-se usando o HiSLIP
ins_ngm202 = resource_manager.open_resource("TCPIP0::192.168.80.18::hislip0")
```

Normalmente, o uso do HiSLIP é preferível quando a camada VISA o suporta; no entanto, o uso do VXI-11 é mais comum, especialmente em instrumentos compatíveis com LXI, e os sockets brutos são mais compatíveis com instrumentos mais antigos com capacidade Ethernet anterior ao LXI.

Conexões de tomada

A interface com instrumentos baseados em sockets (ou seja, instrumentos mais antigos ou gateways que não implementam HiSLIP ou VXI-11) **exige mais do que apenas alterar o nome do recurso VISA!** Se você alterar somente o nome do recurso, logo perceberá que qualquer comando de consulta enviado ao instrumento resultará em um tempo limite excedido. Isso ocorre porque é necessário declarar **explicitamente** os caracteres de terminação a serem usados em uma instância de conexão de socket. Consultar o manual ou realizar alguns testes pode ser necessário.

```
ins_k2110 = resource_manager.open_resource('TCPIP0::192.168.80.61::5025::SOCKET')
ins_k2110.read_termination = "\n"
```

Por exemplo, ao usar meu Keithley 2110 através da minha ponte USB-TMC para Ethernet , a configuração adicional necessária é o valor *read_termination* .

```
ins_hmp4040 = resource_manager.open_resource("TCPIP0::192.168.80.13::5025::SOCKET")
ins_hmp4040.read_termination = "\n"
ins_hmp4040.query_termination = "\n"
```

No entanto, alguns instrumentos são um pouco diferentes – por exemplo, meu Rohde & Schwarz HMP4040 decide responder com um caractere de nova linha **para um comando de configuração e um comando de**

consulta . Como resultado, ele precisa que tanto *o `read_termination`* quanto *o `query_termination`* estejam configurados.

Instrumentos não confiáveis/difíceis de usar

Infelizmente, alguns instrumentos podem não ser confiáveis quando operados remotamente. Isso pode resultar na perda de controle do instrumento durante uma sequência de testes, em bipes contínuos ou na exibição de erros aleatórios e enigmáticos após um certo período de funcionamento, ou ainda em comandos que não funcionam conforme o descrito no manual.

Antes de se desesperar, é bom verificar se você conectou o instrumento usando um cabo de qualidade. Já tive casos em que um cabo USB ruim, um hub defeituoso ou problemas de compatibilidade entre o controlador USB do meu computador e o instrumento de teste causavam erros genéricos aleatórios e não reproduzíveis. Às vezes, trocar o cabo por um cabo blindado de qualidade e com o comprimento mínimo necessário pode resolver o problema.

Além disso, pode ser necessário reiniciar seus instrumentos ocasionalmente para garantir que permaneçam confiáveis. A substituição dos instrumentos é outra opção, embora seja uma opção cara.

Mas, às vezes, é importante **programar de forma defensiva** e tentar se recuperar de erros da melhor maneira possível. Isso pode incluir o uso de loops *`try`* e *`except`* para tentar novamente as leituras várias vezes caso ocorra um erro. Talvez, após um certo número de erros, você possa tentar fechar o identificador do instrumento e restabelecer a conexão.

Na falta de outra solução, cheguei ao ponto de integrar um código para acionar interruptores de energia controláveis remotamente, reiniciando os instrumentos à força para que voltem a funcionar, embora essa seja uma abordagem de "força bruta". Felizmente, a maioria dos instrumentos modernos tende a ser bastante confiável.

A automação pode ser perigosa!

É importante lembrar: a automação é conveniente e poderosa, mas também pode ser extremamente perigosa. Imagine um cenário em que você esteja carregando uma bateria com uma fonte de alimentação — o que acontece se, de repente, você perder a conexão com o seu instrumento ou se ele parar de responder completamente? Talvez a bateria exploda, pegue fogo, machuque alguém ou incendeie sua casa.

Embora muitos testes possam ser realizados de forma prática com automação, é importante fazê-lo com cuidado para garantir que o pior cenário possível não acabe custando caro para você ou para o seu equipamento. Ter um interruptor de energia remoto de reserva para isolar a alimentação do seu rack de instrumentos pode ser um plano de contingência simples, que pode ser acionado por um script caso algo dê errado. Além disso, garantir que seus scripts o alertem sobre problemas e que você tenha alguém disponível para contatar e que saiba como desligar seus instrumentos com segurança pode ser vital. **Seja qual for a sua escolha, não deixe isso para depois.**

Mesmo sem isso, pode ser uma boa ideia ter um segundo instrumento à mão para monitorar o primeiro – por exemplo, em situações onde pode ser crucial obter uma segunda opinião, já que pequenas variações de tensão podem levar a consequências graves. Muitos instrumentos de medição possuem proteções integradas contra sobretensão, sobrecorrente e sobrecarga – configurá-las adequadamente pode fornecer uma segunda camada de segurança, mesmo que seu código configure a saída incorretamente.

Conclusão

A ideia central da arquitetura VISA é permitir que todos os tipos de instrumentos sejam controlados por software de forma unificada. Você não precisa se preocupar se o seu equipamento é de um único fornecedor ou não – tudo deve funcionar perfeitamente. O conjunto de comandos SCPI padroniza o formato e os conjuntos de comandos comuns para diferentes tipos de equipamentos de teste. A maioria dos instrumentos de fornecedores renomados de equipamentos de teste suporta o uso de uma conexão de controle remoto via GPIB, RS-232, USB ou Ethernet e o uso de comandos SCPI.

Espero que, ao ler este tutorial, você perceba que a automação de equipamentos de teste não precisa ser difícil. Com o uso do *pyvisa* como intermediário entre o Python e a camada VISA, é possível controlar instrumentos usando a linguagem Python, fácil de aprender, mantendo-se totalmente gratuito e multiplataforma. O formato padrão dos comandos SCPI, uma vez compreendido, facilita o comando de instrumentos para executar ações e realizar medições. Ao automatizar o uso de equipamentos de teste, você pode realizar experimentos melhores, tornando-os consistentemente repetíveis, obter leituras mais precisas do que faria manualmente e até mesmo realizar experimentos autônomos, nos quais o programa pode tomar decisões com base nas leituras obtidas dos instrumentos. É claro que existem alguns detalhes e boas práticas a serem considerados, alguns dos quais abordei neste tutorial.

Começar não é tão difícil e aqueles manuais de programação volumosos não são motivo para temer. Depois de perceber todas as possibilidades que a automação SCPI oferece, você provavelmente não voltará a mexer em botões e anotar valores na hora de realizar sequências de teste. Então, se você tem um instrumento compatível com SCPI, é hora de conectá-lo e começar a automatizar!

Sobre lui_gough
Sou um apaixonado por eletrônica, informática, fotografia, rádio, satélite e outros hobbies técnicos. Clique para saber mais sobre mim!
Ver todas as publicações de lui_gough →

Este artigo foi publicado em Eletrônica e marcado com as tags faça você mesmo , eletrônica , improvisar , programação , Python , software , teste , equipamento de teste , testado , testando , tutorial . Adicione o link permanente aos seus favoritos .

6 respostas para o *tutorial: Introdução à automação SCPI de equipamentos de teste com pyvisa*

drew2355 *diz:*
28 de março de 2021, às 18h41

Um ótimo artigo, mesmo que seja apenas para explicar o VISA. Interligar instrumentos é uma verdadeira dor de cabeça. Já fiz isso com equipamentos de RF da HP e da R&S.
Drew VK4ZXI

Responder

qwerty *diz:*
6 de maio de 2021, às 19h07

Fiquei muito feliz em encontrar isso. Depois de um ano trabalhando com pyVISA e multímetros Keithley, sinto que ainda não entendi a maioria dos conceitos básicos e é bem difícil encontrar explicações tão acessíveis. Seu artigo me ajudou a entender, muito obrigado por escrevê-lo.

Responder

Pavan *diz:*
28 de fevereiro de 2022, às 21h06

Apresentação impecável, com detalhes minuciosos. Obrigado.

Responder

Nav *diz:*
7 de janeiro de 2023, às 5h26

Artigo muito informativo; obrigado por tê-lo compilado.

Responder

Dave Landis *diz:*
30 de setembro de 2023, às 19h32

Muito obrigado por este artigo. Uso o PyVisa há cerca de um ano e aprendi alguns truques e dicas muito úteis. Estou querendo substituir uma grande biblioteca de programas LabVIEW antigos por código Python e toda dica é bem-vinda. Parabéns pelo excelente trabalho.

Responder

Daniel *diz:*
2 de setembro de 2025, às 12h06

Consegui fazer isso funcionar no meu programa em Python. Foi de enorme ajuda.

Responder