SATCFe Documentation

Release 2.2

Base4 Sistemas EIRELI

jan. 19, 2022

Sumário

A Secretaria da Fazenda do Estado de São Paulo, SEFAZ/SP, implantou através da Portaria CAT 147 o *SAT-CF-e* (Sistema de Autorização e Transmissão de Cupons Fiscais eletrônicos), em substituição às impressoras fiscais (*ECF-IF*).

Esta documentação diz respeito ao projeto SATCFe desenvolvido pela Base4 Sistemas com o objetivo de abstrair o acesso ao Equipamento SAT através da linguagem Python, tornando trivial o acesso às funções da biblioteca SAT, resultando em respostas prontas para serem utilizadas pela aplicação cliente, normalmente um software de ponto-de-venda (PDV).

A figura abaixo ilustra a topologia básica do SAT-CF-e no estabelecimento comercial. Em uma operação típica, o aplicativo comercial envia o CF-e de venda para o equipamento SAT que irá completar, validar, assinar e transmitir o documento para a SEFAZ para autorização. Se o documento for autorizado ele será devolvido para o aplicativo comercial que irá emitir o extrato do CF-e para o consumidor.

Na maioria das vezes, a aplicação cliente acessa as funções da biblioteca SAT para transmitir à SEFAZ os dados de uma venda, o CF-e de venda, ou para cancelar o último CF-e transmitido. Além disso, a biblioteca SAT contém várias outras funções que possibilitam a execução de tarefas administrativas e de configurações do equipamento SAT. Todas essas funções serão detalhadas uma-a-uma nesta documentação.

CAPÍTULO 1

Projetos Relacionados

Este projeto é apenas uma parte de um total de cinco projetos que compõem uma solução compreensível para a tecnologia SAT-CF-e em linguagem Python, disponíveis para integração nas aplicações de ponto-de-venda. São eles:

- **Projeto SATComum** Mantém o código que é compartilhado pelos outros projetos relacionados, tais como validação, formatação e valores constantes.
- **Projeto SATExtrato** Impressão dos extratos do CF-e-SAT. Este projeto é capaz de imprimir extratos de documentos de venda ou de cancelamento diretamente a partir dos documentos XML que os representam. A impressão tem um alto grau de compatibilidade com mini-impressoras (conhecidas como impressoras não-fiscais) já que é baseada na tecnologia Epson® ESC/POS™ através do projeto **PyESCPOS**.
- **Projeto SATHub** Torna possível o compartilhamento de equipamentos SAT com múltiplos pontos- de-venda, além de tornar possível que aplicações heterogêneas, escritas em outras linguagens de programação ou de outras plataformas, acessem o equipamento SAT.
- **Projeto PyESCPOS** Implementa o suporte à tecnologia Epson® ESC/POS™ compatível com a imensa maioria das mini-impressoras disponíveis no mercado.

CAPÍTULO 2

Participe

Participe deste projeto ou de qualquer um dos projetos relacionados. Se você puder contribuir com código, excelente! Faça um clone do repositório, modifique o que acha que deve e faça o *pull-request*. Teremos prazer em aceitar o seu código.

Se você não quer (ou não pode) programar, também pode contribuir com documentação. Ou ainda, se você vir algo errado ou achar que algo não está certo, conte pra gente criando um incidente na página do projeto.

Siga-nos no Github ou no Twitter.

CAPÍTULO 3

Conteúdo

3.1 Configuração Básica

Existem dois cenários para configuração de um Cliente SAT. No primeiro, o equipamento SAT está conectado diretamente ao computador em que o aplicativo comercial está instalado. No segundo, o aplicativo comercial compartilha o equipamento SAT com outros aplicativos através de uma rede local.

No primeiro cenário, basta instanciar um *ClienteSATLocal* e configurar o acesso à biblioteca SAT e o código de ativação:

```
from satcfe import BibliotecaSAT
from satcfe import ClienteSATLocal

cliente = ClienteSATLocal(BibliotecaSAT('/opt/fabricante/libsat.so'),
        codigo_ativacao='12345678')
```

No segundo cenário, basta instanciar um *ClienteSATHub* e apontá-lo para o servidor SATHub. Note que neste caso, será necessário informar o **número do caixa**, para que o SATHub possa determinar a origem das solicitações.

```
from satcfe import ClienteSATHub

cliente = ClienteSATHub('192.168.0.101', 8088, numero_caixa=15)
```

Em qualquer cenário, depois de instanciado o cliente, o acesso às funções SAT é absolutamente idêntico:

```
resposta = cliente.consultar_sat()
```

Nota: A maneira como essas configurações serão persistidas pela aplicação comercial e como elas serão atribuídas na iniciação da aplicação está fora do escopo deste projeto.

3.2 Instanciando um Cliente SAT

Para ter acesso às funções SAT é preciso instanciar um *cliente* SAT, que pode ser um **Cliente Local**, no cenário em que o equipamento SAT está conectado ao mesmo computador em que está instalado o aplicativo comercial, ou um **Cliente SATHub**, quando o acesso ao equipamento SAT é compartilhado.

3.2.1 Cliente Local

Em um cliente local o acesso ao equipamento SAT é feito através da biblioteca SAT que é fornecida pelo fabricante do equipamento, distribuída normalmente como uma DLL (*dinamic-link library*, `.dll`) ou *shared library* (`.so`), de modo que é necessário indicar o caminho completo para a biblioteca.

```
from satcfe import BibliotecaSAT
from satcfe import ClienteSATLocal

cliente = ClienteSATLocal(BibliotecaSAT('/opt/fabricante/libsat.so'),
        codigo_ativacao='12345678')

resposta = cliente.consultar_sat()
```

3.2.2 Cliente SATHub

Em um cliente SATHub o acesso ao equipamento SAT é compartilhado e feito através de uma requisição HTTP para endereço onde o servidor SATHub responde. Em ambos os casos a chamada à função é exatamente a mesma, com exceção da instanciação do cliente:

```
from satcfe import ClienteSATHub

cliente = ClienteSATHub('192.168.0.101', 8088, numero_caixa=7)
resposta = cliente.consultar_sat()
```

Via de regra o código que acessa as funções da biblioteca SAT não deveria se importar se o cliente é um cliente local ou remoto, de modo que o aplicativo comercial precisa apenas implementar um *factory* que resulte no cliente SAT adequadamente configurado.

3.2.3 Numeração de Sessões

Um outro aspecto relevante é a questão da numeração de sessões, que conforme a ER SAT, item 6, alínea "d", diz o seguinte:

> O SAT deverá responder às requisições do AC de acordo com o número de sessão recebido. O aplicativo comercial deverá gerar um número de sessão aleatório de 6 dígitos que não se repita nas últimas 100 comunicações.

Para um cliente SAT local, é fornecida uma implementação básica de numeração de sessão que é encontrada na classe *`satcfe.base.NumeroSessaoMemoria`*, que é capaz de atender o requisito conforme descrito na ER SAT. Entretando, essa implementação básica não é capaz (ainda) de persistir os números gerados.

Se for necessário utilizar um esquema de numeração de sessão diferente, basta escrever um e passá-lo como argumento durante a criação do cliente local. Um numerador de sessão é apenas um *callable* que, quando invocado, resulta no próximo número de sessão a ser usado em uma função SAT. Por exemplo:

```
def meu_numerador():
    numero = ... # lógica diferente
    return numero

cliente = ClienteSATLocal(
        BibliotecaSAT('/opt/fabricante/libsat.so'),
        codigo_ativacao='12345678'
        numerador_sessao=meu_numerador)
```

Para os clientes SATHub há um esquema de numeração de sessão mais robusto, já que as requisições tem origem em caixas (pontos-de-venda) diferentes, o requisito é resolvido de maneira a evitar colisões de numeração ou repetição de numeração mesmo atendendo requisições concorrentes. Consulte a documentação do projeto SATHub para os detalhes.

3.3 Funções SAT

O acesso às funções SAT se dá através de uma biblioteca que é fornecida pelo fabricante do equipamento SAT. Este projeto abstrai o acesso à essa biblioteca tornando possível acessar um equipamento SAT conectado no computador local ou compartilhar um equipamento SAT entre dois ou mais computadores, acessando-o remotamente via API RESTful.

Se você estiver acessando um equipamento SAT conectado ao computador local, então deverá usar um `ClienteSATLocal`, cuja configuração já foi discutida em *Configuração Básica*.

Se estiver compartilhando um equipamento SAT, então deverá usar um `ClienteSATHub`, cuja configuração também já foi demonstrada.

Nota: Instalar um servidor SATHub está fora do escopo desta documentação. Consulte a documentação do projeto SATHub para saber como instalar e configurar um servidor SATHub em desenvolvimento ou em produção.

Uma vez configurado o cliente SAT, basta invocar os métodos correspodentes às funções SAT, que serão demonstradas mais adiante nesta documentação.

Nota: Sobre os nomes dos métodos Os nomes das funções SAT neste projeto foram modificados dos nomes originais para ficarem compatíveis com o estilo de código Python para nomes de métodos, funções, etc. Mas a modificação é simples e segue uma regra fácil de converter de cabeça. Por exemplo:

```
ComunicarCertificadoICPBRASIL  ->  comunicar_certificado_icpbrasil
TesteFimAFim                   ->  teste_fim_a_fim
```

As palavras são separadas por um caracter de sublinha e o nome é todo convertido para letras minúsculas.

3.3.1 Lidando com as Respostas

As respostas contém os atributos que são descritos na ER SAT com nomes que sejam o mais próximo possível da descrição oficial. Por exemplo, a função `ConsultarSAT` está descrita na ER SAT no item 6.1.5 e os detalhes da resposta à esta função estão descritos no item 6.1.5.2 e diz o seguinte:

> **Retorno** "`numeroSessao|EEEEE|mensagem|cod|mensagemSEFAZ`"

Dessa forma, a resposta à função `ConsultarSAT` deverá conter atributos com os mesmos nomes descritos na ER SAT:

```
resposta = cliente.consultar_sat()
print(resposta.numeroSessao)  # resulta em 'int'
print(resposta.EEEEE)  # resulta em 'unicode'
print(resposta.mensagem)
print(resposta.cod)
print(resposta.mensagemSEFAZ)
```

Caso ocorra um erro ao invocar o método `consultar_sat()` será lançada uma exceção `ExcecaoRespostaSAT` contendo os detalhes do problema.

3.3.2 Lidando com Exceções

Quando uma função é invocada, seja através de um *`ClienteSATLocal`* ou *`ClienteSATHub`*, existem duas exceções principais que podem ocorrer: *`ErroRespostaSATInvalida`* ou *`ExcecaoRespostaSAT`*.

Quando a exceção `ErroRespostaSATInvalida` é levantada, significa que a resposta retornada pelo equipamento SAT não está em conformidade com a ER SAT, geralmente por que a biblioteca resultou em uma sequência que não possui os elementos que deveriam estar presentes, seja uma resposta de sucesso na execução da função ou não.

Por outro lado será comum lidar com `ExcecaoRespostaSAT`. Esta exceção indica que a comunicação entre a AC e o equipamento correu bem mas execução da função não obteve êxito. É o caso quando invocar a função `ConsultarSAT` e o equipamento estiver ocupado processando uma outra coisa; a exceção poderá indicar o erro, já que ela contém uma resposta:

```
>>> # suponha que o equipamento SAT está ocupado
>>> import sys
>>> resposta = cliente.consultar_sat()
Traceback (most recent call last):
  ...
ExcecaoRespostaSAT: ConsultarSAT, numeroSessao=567192, EEEEE='08098', mensagem="SAT␣
↪em processamento. Tente novamente.", cod="", mensagemSEFAZ=""

>>> resposta = sys.last_value
>>> resposta.mensagem
'SAT em processamento. Tente novamente.'

>>> resposta.EEEEE
'08098'

>>> resposta.numeroSessao
567192
```

O truque acima foi obter o objeto da exceção levantada de `sys.last_value`, que é similar ao que deveria fazer no bloco de tratamento da exceção `ExcecaoRespostaSAT`, por exemplo:

```
try:
    resposta = cliente.consultar_sat()
    # faz algo com a resposta...

except ErroRespostaSATInvalida as ex_resp_invalida:
    # exibe o erro para o operador...
    break

except ExcecaoRespostaSAT as ex_resposta:
```

(continues on next page)

(continuação da página anterior)

```
    resposta = ex_resposta.resposta
    if resposta.EEEEE == '08098':
        # o equipamento SAT está ocupado
        # pergunta ao operador de caixa se quer tentar novamente...
        pass
```

Obviamente, muita coisa pode dar errado entre o aplicativo comercial e a SEFAZ, então utilize a regra básica de tratamento de exceções recomendada, mantendo uma cláusula `except` de *fallback*, por exemplo:

```
try:
    resposta = cliente.enviar_dados_venda(cfe)
    # faz algo com a resposta aqui

except ErroRespostaSATInvalida as ex_sat_invalida:
    # o equipamento retornou uma resposta que não faz sentido;
    # loga, e lança novamente ou lida de alguma maneira
    pass

except ExcecaoRespostaSAT as ex_resposta:
    # o equipamento retornou mas a função não foi bem sucedida;
    # analise 'EEEEE' para decidir o que pode ser feito
    pass

except Exception as ex:
    # uma outra coisa aconteceu
    pass
```

Aviso: Evite silenciar (ignorar) exceções. Se não sabe o porquê, veja o tópico sobre Tratamento de Exceções no tutorial de Python.

3.3.3 Funções Básicas e de Consulta

Estas são provavelmente as funções mais básicas da biblioteca SAT. São aquelas funções que normalmente são as primeiras a serem invocadas quando se está iniciando o procedimento de integração do SAT com o aplicativo comercial. Os exemplos dizem respeito a qualquer cliente SAT, local ou via SATHub.

A maioria das funções SAT resulta em uma resposta padrão no estilo:

```
numeroSessao|EEEEE|mensagem|cod|mensagemSEFAZ
```

Portanto, os atributos `numeroSessao`, `EEEEE`, `mensagem`, `cod` e `mensagemSEFAZ` estarão disponíveis na maioria das respostas, conforme visto em *Lidando com as Respostas*:

ConsultarSAT

A função `ConsultarSAT` (ER item 6.1.5, método *`consultar_sat()`*) é usada para testar a comunicação com o equipamento SAT. Seu uso é simples e direto e, se nenhuma exceção for lançada, é seguro acessar os atributos da resposta conforme esperado.

```
>>> resp = cliente.consultar_sat()
>>> resp.mensagem
'SAT em Operação'
```

ConsultarStatusOperacional

A função `ConsultarStatusOperacional` (ER item 6.1.7, método *`consultar_status_operacional()`*) retorna atributos que mostram diversas informações a respeito do equipamento SAT. A resposta para esta função é direta e simples, mas se você verificar a documentação da ER SAT pode ficar confuso quanto aos atributos da resposta. A ER SAT diz que o retorno da função é:

```
numeroSessao|EEEEE|mensagem|cod|mensagemSEFAZ|ConteudoRetorno
```

Entretando, a resposta **não possui** um atributo `ConteudoRetorno`, por que ele se expande em outros atributos que são documentados na ER SAT em uma tabela separada. É como se o retorno fosse:

```
numeroSessao|EEEEE|mensagem|cod|mensagemSEFAZ|NSERIE|TIPO_LAN|LAN_IP|...
```

Por exemplo:

```
>>> resp = cliente.consultar_status_operacional()
>>> resp.mensagem
'Resposta com Sucesso'

>>> resp.NSERIE
320008889

>>> resp.STATUS_LAN
'CONECTADO'

>>> resp.DH_ATUAL
datetime.datetime(2015, 6, 25, 15, 26, 37)
```

ConsultarNumeroSessao

A função `ConsultarNumeroSessao` (ER item 6.1.8, método *`consultar_numero_sessao()`*) permite consultar a resposta para sessão executada anteriormente. Essa função é especial no sentido de que sua resposta será a resposta para a função executada na sessão que está sendo consultada.

Por exemplo, suponha que a última sessão executada seja um cancelamento, com número de sessão `555810`. Se este número de sessão for consultado, a resposta será a resposta de um cancelamento, resultando em uma instância de *`RespostaCancelarUltimaVenda`*.

```
>>> resp = cliente.consultar_numero_sessao(555810)
>>> resp
<satcfe.resposta.cancelarultimavenda.RespostaCancelarUltimaVenda at 0x7ffb171e02d0>
```

Nota: A documentação não deixa claro, mas os testes executados contra três equipamentos SAT de fabricantes diferentes se comportaram da seguinte maneira:

- Apenas a sessão executada imediatamente antes é que será considerada, ou seja, não adianta especificar uma sessão que tenha sido processada há duas ou mais sessões anteriores;
- Se a última sessão executada for de uma função de consulta de número de sessão (algo como uma *meta consulta*), a função também irá falhar.

ConsultarUltimaSessaoFiscal

A função `ConsultarUltimaSessaoFiscal` (ER item 6.1.16, método *`consultar_ultima_sessao_fiscal()`*), como o nome sugere, resulta na resposta da última sessão fiscal executada pelo equipamento SAT. É considerada uma "sessão fiscal" um comando de venda ou de cancelamento de venda, respectivamente *`enviar_dados_venda()`* ou *`cancelar_ultima_venda()`*, e suas respostas, *`RespostaEnviarDadosVenda`* ou *`RespostaCancelarUltimaVenda`*.

Por exemplo, suponha que a última sessão fiscal executada pelo equipamento SAT tenha sido um comando de venda:

```
>>> resp = cliente.consultar_ultima_sessao_fiscal()
>>> resp
<satcfe.resposta.enviardadosvenda.RespostaEnviarDadosVenda at 0x7f1817971950>
```

Conforme descrito na especificação de requisitos do SAT, se o equipamento ainda não tiver executado nenhum comando fiscal (venda ou cancelamento), a resposta deverá indicar o código de retorno `EEEEE` igual a `19003` que significa "Não existe sessão fiscal".

```
>>> try:
...     # suponha que o equipamento nunca tenha executado um comando fiscal
...     resp = cliente.consultar_ultima_sessao_fiscal()
... except ExcecaoRespostaSAT as err:
...     pass
...

>>> err.resposta.EEEEE
'19003'

>>> err.resposta.mensagem
'Não existe sessão fiscal'
```

Veja mais detalhes em *Venda e Cancelamento*.

Novo na versão 2.0.

ExtrairLogs

A função `ExtrairLogs` (ER item 6.1.12, método *`extrair_logs()`*) retorna os registros de log do equipamento SAT. A resposta para esta função possui duas particularidades: primeiro que os registros de log podem ser automaticamente decodificados através do método *`conteudo()`*; segundo que o nome dado para este campo pela ER SAT fica muito longo e, portanto, foi chamado apenas de `arquivoLog`.

```
>>> resp = cliente.extrair_logs()
>>> resp.mensagem
'Transferência completa'

>>> resp.arquivoLog
'MjAxNTA2MTIxNTAzNTB...jaGF2ZXMgZW5jb250cmFkbyBubyB0b2tlbg=='

>>> print(resp.conteudo())
20150612150350|SAT|info|nvl 2:token inicializado
20150612150350|SAT|info|nvl 2:par de chaves encontrado no token
20150612150350|SAT|info|nvl 2:certificado encontrado no token
20150612150350|SAT-SEFAZ|info|nvl 2:(CFeStatus) acessado o webservice
20150612150351|SAT|erro|nvl 0:(no error) marca inicio dos logs (01.00.00:48)
20150612150351|SAT|info|nvl 1:Equipamento inicializado
```

(continues on next page)

(continuação da página anterior)

```
20150612150352|SEFAZ-SAT|info|nvl 2:(CFeStatus) status do equipamento recebido pela␣
↪SEFAZ
20150612150356|SAT|info|nvl 1:relogio sincronizado com sucesso
20150612150356|SAT-SEFAZ|info|nvl 2:(CFeComandos) acessado o webservice
20150612153407|SEFAZ-SAT|info|nvl 2:(CFeComandos) não existem comandos pendentes
20150612153544|AC-SAT|info|nvl 2:recebida mensagem referente a função ConsultarSAT
20150612153544|SAT-AC|info|nvl 2:enviando mensagem referente a função ConsultarSAT
20150612153544|AC-SAT|info|nvl 2:recebida mensagem referente a função␣
↪ConsultarStatusOperacional
```

Também é possível salvar o conteúdo decodificado dos registros de log através do método *salvar()*:

```
>>> resp = cliente.extrair_logs()
>>> resp.salvar()
'/tmp/tmpNhVSHi-sat.log'
```

3.3.4 Funções de Configuração/Modificação

As funções a seguir são utilizadas para configurar o equipamento SAT ou acabam por modificar certos registros de informações que ficam permanentemente gravadas no equipamento.

AtivarSAT

A função `AtivarSAT` (ER item 6.1.1, método *ativar_sat()*) é usada para realizar a ativação do equipamento SAT tornando-o apto para realizar vendas e cancelamentos. Para maiores detalhes consulte o item 2.1.1 da ER SAT.

```
>>> from satcomum import constantes
>>> from satcomum import br
>>> cnpj_contribuinte = '12345678000199'
>>> resp = cliente.ativar_sat(constantes.CERTIFICADO_ACSAT_SEFAZ,
...          cnpj_contribuinte, br.codigo_ibge_uf('SP'))
...
>>> resp.csr()
'-----BEGIN CERTIFICATE REQUEST-----
MIIBnTCCAQYCAQAwXTELMAkGA1UEBhMCU0cxETAPBgNVBAoTCE0yQ3J5cHRvMRIw
 ...
9rsQkRc9Urv9mRBIsredGnYECNeRaK5R1yzpOowninXC
-----END CERTIFICATE REQUEST-----
```

Atenção: É nesta função que é definido o **Código de Ativação** do equipamento SAT.

Este código é uma senha que é enviada ao equipamento a cada função executada. **Se o equipamento ainda não estiver ativo**, esta deverá ser a primeira função a ser executada e o código de ativação *informado ao instanciar o cliente SAT* é o código que será usado para definir o código de ativação do equipamento.

Veja *TrocarCodigoDeAtivacao* para outros detalhes.

ComunicarCertificadoICPBRASIL

A função `ComunicarCertificadoICPBRASIL` (ER item 6.1.2, método *comunicar_certificado_icpbrasil()*) é complementar à função `AtivarSAT` e é usada para enviar

à SEFAZ o conteúdo do certificado emitido pela ICP Brasil.

```
>>> with open('certificado.pem', 'r') as f:
...     certificado = f.read()
...
>>> resp = cliente.comunicar_certificado_icpbrasil(certificado)
>>> resp.mensagem
'Certificado transmitido com sucesso'
```

ConfigurarInterfaceDeRede

A função `ConfigurarInterfaceDeRede` (ER item 6.1.9, método *`configurar_interface_de_rede()`*) é utilizada para configurar o acesso à rede para que o equipamento SAT possa ter acesso à internet. Os parâmetros de configuração são informados através de uma instância da classe *`ConfiguracaoRede`*.

Nota: Se o equipamento ainda não tiver sido ativado, o código de ativação ao invocar esta função deverá ser `00000000` (oito dígitos zero).

```
>>> from satcomum import constantes
>>> from satcfe.rede import ConfiguracaoRede
>>> rede = ConfiguracaoRede(
...         tipoInter=constantes.REDE_TIPOINTER_ETHE,
...         tipoLan=constantes.REDE_TIPOLAN_DHCP)
...
>>> resp = cliente.configurar_interface_de_rede(rede)
>>> resp.mensagem
'Rede configurada com sucesso'
```

AssociarAssinatura

A função `AssociarAssinatura` (ER item 6.1.10, método *`associar_assinatura()`*) é usada para vincular a assinatura do aplicativo comercial ao equipamento SAT. Essa mesma assinatura é utilizada no atributo `signAC` ao realizar *vendas* e cancelamentos.

```
>>> resp = cliente.associar_assinatura(
...         '11111111111111222222222222222',
...         'RVlHYkYzcytsZFdiekM4SExmNFVLaXlaZF...')
...
>>> resp.mensagem
'Assinatura do AC registrada'
```

O primeiro argumento, `sequencia_cnpj`, deve ser uma string de 28 digitos contendo o CNPJ da software house e o CNPJ do estabelecimento contribuinte.

O segundo argumento, `assinatura_ac`, deve ser uma sequência de 344 caracteres, contendo o hash SHA256 codificado em Base64.

Gerando a Assinatura AC

Este exemplo demonstra como gerar a assinatura em um terminal Linux, usando OpenSSL e a parte privada da sua chave RSA (assumindo que a chave privada tenha sido extraída do e-CNPJ ou arquivo `pfx` do certificado digital da software house em um arquivo chamado `~/.keys/private.pem`):

```
$ echo -n 11111111111111222222222222222 | \
    openssl dgst -sha256 -sign ~/.keys/private.pem | \
    openssl enc -base64 -e
```

A saída desse comando é o valor que deverá ser informado no argumento `assinatura_ac`.

Dica: Consulte o item 2.1.3 da ER SAT para conhecer a especificação.

AtualizarSoftwareSAT

A função `AtualizarSoftwareSAT` (ER item 6.1.11, método *`atualizar_software_sat()`*) é usada para atualização do software básico do equipamento SAT.

```
>>> resp = cliente.atualizar_software_sat()
>>> resp.mensagem
'Software atualizado com sucesso'
```

BloquearSAT

A função `BloquearSAT` (ER item 6.1.13, método *`bloquear_sat()`*) é usada para realizar o bloqueio operacional do equipamento SAT.

```
>>> resp = cliente.bloquear_sat()
>>> resp.mensagem
'Equipamento SAT bloqueado com sucesso'
```

DesbloquearSAT

A função `DesbloquearSAT` (ER item 6.1.14, método *`desbloquear_sat()`*) é usada para realizar o desbloqueio operacional do equipamento SAT.

```
>>> resp = cliente.desbloquear_sat()
>>> resp.mensagem
'Equipamento SAT desbloqueado com sucesso'
```

TrocarCodigoDeAtivacao

A função `TrocarCodigoDeAtivacao` (ER item 6.1.15, método *`trocar_codigo_de_ativacao()`*) é usada, como o nome sugere, para trocar o código de ativação do equipamento SAT que, na prática, é uma senha que é enviada ao equipamento SAT a cada comando.

```
>>> novo_codigo = 's3cr3t0'
>>> resp = cliente.trocar_codigo_de_ativacao(novo_codigo)
>>> resp.mensagem
'Código de ativação alterado com sucesso'
```

Todo equipamento SAT possui um **código de ativação de emergência**, que acompanha o produto (pode estar escrito no manual do usuário ou em alguma etiqueta na embalagem ou no próprio equipamento). Caso o código de ativação seja perdido, é possível trocar o código de ativação usando o código de ativação de emergência:

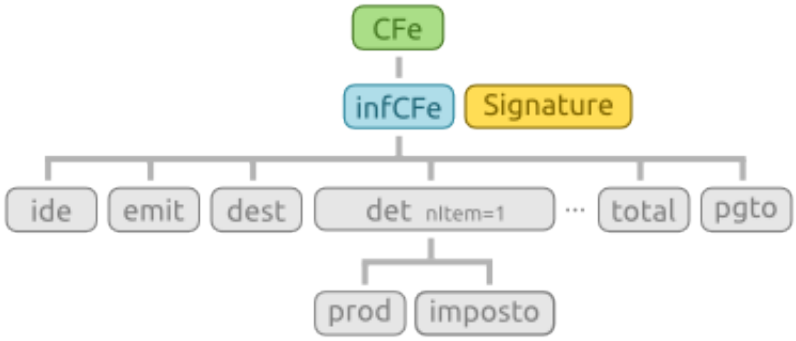
```
>>> from satcomum import constantes
>>> novo_codigo = 's3cr3t0'
>>> resp = cliente.trocar_codigo_de_ativacao(
...          novo_codigo,
...          opcao=constantes.CODIGO_ATIVACAO_EMERGENCIA,
...          codigo_emergencia='d35c0nh3c1d0')
...
>>> resp.mensagem
'Código de ativação alterado com sucesso'
```

Veja *AtivarSAT* para mais informações.

3.4 Venda e Cancelamento

3.4.1 Anatomia do CF-e

O *Cupom Fiscal eletrônico*, CF-e, é um documento fiscal com validade jurídica que não existe fisicamente, mas apenas de forma eletrônica, em formato XML, que descreve todos os aspectos práticos de uma operação de venda ou do cancelamento de uma venda. A figura abaixo ilustra a anatomia de um CF-e de venda, destacando todos os seus elementos de mais alto nível.



Um documento CF-e que se compare com a ilustração, grosseiramente se traduz para o seguinte fragmento XML:

```
<CFe>
  <infCFe>
    <ide/>
    <emit/>
    <dest/>
    <det nItem="1">
      <prod/>
      <imposto/>
    </det>
    <total/>
    <pgto/>
  </infCFe>
  <Signature/>
</CFe>
```

O ponto central da tecnologia SAT-CF-e, do ponto de vista do desenvolvedor do aplicativo comercial, é o modelo através do qual um CF-e é construído até se transformar em um documento com validade jurídica.

1. O aplicativo comercial inicia o CF-e construindo a maior parte dos elementos a partir dos dados da venda e o envia para o equipamento SAT através da função `EnviarDadosVenda`;
2. O equipamento SAT complementa o CF-e, calculando e incluindo outras informações que são de sua responsabilidade e assinando digitalmente o documento, e o transmite para a SEFAZ;
3. A SEFAZ valida o documento e o retorna para o equipamento SAT que, finalmente, retorna a resposta para o aplicativo comercial.

Para compor um CF-e o desenvolvedor do aplicativo comercial deverá observar a coluna **Origem** da tabela que descreve os elementos do CF-e nos itens **4.2.2** (*layout do arquivo de venda*) e **4.2.3** (*layout do arquivo de cancelamento*). Os elementos onde a coluna **Origem** indicar `AC` são os elementos que o aplicativo comercial deverá incluir no XML. Os elementos indicados com `SAT` são os elementos que o equipamento SAT deverá incluir.

4.2.2. Leiaute do arquivo de Venda (CF-e-SAT)

O leiaute do arquivo de venda (arquivo CF-e-SAT) que será gerado pelo SAT deve

Origem	#	ID	Campo	Descrição	Elemento	Pai	Tipo	Ocorr.
AC		-	CFe	TAG raiz do CF-e	G	-		1-1
A - Dados do Cupom Fiscal Eletrônico								
Origem	#	ID	Campo	Descrição	Elemento	Pai	Tipo	Ocorr.
AC		A01	infCFe	Grupo das informações do CF-e	G	Raiz	-	1-1
SAT		A02	Versao	Versão do leiaute do CF-e	A	A01	N	1-1
AC		A03	versaoDadosEnt	Versão do leiaute do arquivo de	A	A01	N	1-1

3.4.2 Entidades

No contexto deste projeto as **Entidades** são as classes que são utilizadas para descrever uma venda ou um cancelamento. A documentação da API contém uma tabela que relaciona as classes de entidades aos elementos XML descritos na ER SAT, em *Módulo satcfe.entidades*.

Lidar com API de entidades não é difícil. O exemplo abaixo mostra uma sessão do interpretador onde é criado uma instância de *`LocalEntrega`* totalmente inválida:

```
>>> from satcfe.entidades import LocalEntrega
>>> entrega = LocalEntrega()
>>> entrega.validar()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "satcfe/entidades.py", line 298, in validar
    'atributos invalidos.'.format(self.__class__.__name__))
cerberus.cerberus.ValidationError: Entidade "LocalEntrega" possui atributos invalidos.
>>> entrega.erros
{'xBairro': 'required field', 'nro': 'required field', 'UF': 'required field', 'xMun
↪': 'required field', 'xLgr': 'required field'}
```

Para obter o fragmento XML de uma entidade, faça:

```
>>> entrega = LocalEntrega(
...          xLgr='Rua Armando Gulim',
...          nro='65',
...          xBairro=u'Parque Glória III',
...          xMun='Catanduva',
...          UF='SP')
>>> entrega.documento(incluir_xml_decl=False)
'<entrega><xLgr>Rua Armando Gulim</xLgr><nro>65</nro><xBairro>Parque Gloria III</
↪xBairro><xMun>Catanduva</xMun><UF>SP</UF></entrega>'
```

3.4.3 Criando um CF-e de Venda

Criar um CF-e de venda é simples no que diz respeito à composição dos elementos. Obviamente, no contexto da aplicação comercial, inúmeras outras complexidades se apresentam. Mas este exemplo é capaz de produzir um XML que poderá ser enviado para o equipamento SAT.

Nota: Equipamentos SAT em desenvolvimento podem requerer que os dados do emitente sejam certos dados específicos, bem como o CNPJ que identifica a software house que desenvolve a AC. Consulte a documentação técnica do fabricante do seu equipamento SAT.

```
from satcomum import constantes
from satcfe.entidades import Emitente
from satcfe.entidades import Destinatario
from satcfe.entidades import LocalEntrega
from satcfe.entidades import Detalhamento
from satcfe.entidades import ProdutoServico
from satcfe.entidades import Imposto
from satcfe.entidades import ICMSSN102
from satcfe.entidades import PISSN
from satcfe.entidades import COFINSSN
from satcfe.entidades import MeioPagamento

cfe = CFeVenda(
        CNPJ='08427847000169',
        signAC=constantes.ASSINATURA_AC_TESTE,
        numeroCaixa=2,
        emitente=Emitente(
                CNPJ='61099008000141',
                IE='111111111111'),
        destinatario=Destinatario(
                CPF='11122233396',
                xNome=u'João de Teste'),
        entrega=LocalEntrega(
                xLgr='Rua Armando Gulim',
                nro='65',
                xBairro=u'Parque Glória III',
                xMun='Catanduva',
                UF='SP'),
        detalhamentos=[
                Detalhamento(
                        produto=ProdutoServico(
                                cProd='123456',
                                xProd='BORRACHA STAEDTLER pvc-free',
```

(continues on next page)

(continuação da página anterior)

```
                                CFOP='5102',
                                uCom='UN',
                                qCom=Decimal('1.0000'),
                                vUnCom=Decimal('5.75'),
                                indRegra='A'),
                        imposto=Imposto(
                                icms=ICMSSN102(Orig='2', CSOSN='500'),
                                pis=PISSN(CST='49'),
                                cofins=COFINSSN(CST='49'))),
            ],
        pagamentos=[
                MeioPagamento(
                        cMP=constantes.WA03_DINHEIRO,
                        vMP=Decimal('10.00')),
            ])
```

O XML produzido por este código é um documento CF-e ainda incompleto, que deverá ser enviado ao equipamento SAT pra que seja completado, assinado e transmitido para a SEFAZ. Você poderá ver um exemplo do documento XML gerado por esse código em *XML do CF-e de Venda*.

Ao submeter o CF-e ao equipamento SAT, a resposta será uma instância de `RespostaEnviarDadosVenda` e a partir dela você poderá obter o XML do CF-e-SAT assinado e autorizado, obter os dados para geração do QRCode e outras informações:

```
>>> resposta = cliente.enviar_dados_venda(cfe)
>>> resposta.xml()
u'<?xml version="1.0"?><CFe><infCFe Id="CFe35150761...</Signature></CFe>'

>>> resposta.qrcode()
u'35150761099008000141599000026310000100500297|20150709172317|...JI2BCucA=='

>>> resposta.valorTotalCFe
Decimal('5.75')
```

Dica: Em produção, o atributo `signAC` deverá ser uma sequência resultante da codificação em Base64 do hash SHA256 gerado a partir do CNPJ da software house e do CNPJ do estabelecimento emitente.

A documentação da função *AssociarAssinatura* tem mais detalhes sobre isto.

3.4.4 Criando um CF-e de Cancelamento

Para realizar o cancelamento de um CF-e-SAT de venda, você irá precisar da chave de acesso do documento a ser cancelado:

```
from satcomum import constantes
from satcfe import BibliotecaSAT
from satcfe import ClienteSATLocal
from satcfe.entidades import CFeCancelamento

chave_acesso_venda = ... # obter a chave de acesso do CF-e-SAT de venda

cfecanc = CFeCancelamento(
        chCanc=chave_acesso_venda,
```

(continues on next page)

(continuação da página anterior)

```
        CNPJ='08427847000169',
        signAC=contantes.ASSINATURA_AC_TESTE,
        numeroCaixa=2)

cliente = ClienteSATLocal(BibliotecaSAT('/opt/fabricante/libsat.so'),
        codigo_ativacao='12345678')

resposta = cliente.cancelar_ultima_venda(cfecanc.chCanc, cfecanc)
```

Assim como na venda, o cancelamento irá produzir um XML ainda incompleto que será submetido ao equipamento SAT, que o irá completá-lo, assiná-lo e transmití-lo à SEFAZ. Veja um exemplo do *XML do CF-e de Cancelamento* e do *XML do CF-e-SAT de Cancelamento*.

Se algo der errado durante o cancelamento serão lançadas exceções apropriadas. Mais detalhes em *Lidando com as Respostas* e *Lidando com Exceções*.

Para obter o XML da resposta (o CF-e-SAT de cancelamento) ou os dados do QRcode, use os métodos *xml()* e *qrcode()*.

3.5 Infraestrutura de Alertas

Implementa uma infraestrutura simplificada para checagem de alertas baseados nas informações do status operacional do equipamanto SAT. O objetivo é alertar o operador sobre situações potencialmente problemáticas a fim de que ele tome providências a respeito. Esta infraestrutura permite que os problemas sejam identificados de forma robusta, isolada e extensível. Assim, é possível que com uma única consulta ao status operacional, um número variado de problemas possam ser detectados e explanados com detalhes para o operador do sistema.

Cada alerta é implementado como uma subclasse de *AlertaOperacao* que sobrescreve os métodos *checar()* e *mensagem()*, responsáveis por identificar se o alerta está ativo ou não e em contruir uma mensagem que descreva o status daquele alerta da forma mais clara e detalhada possível, respectivamente.

A intenção é que a checagem dos alertas seja feita de forma automática, quando o sistema (ponto-de-venda) for iniciado, no intuito de que o operador possa resolver a tempo os alertas ativos. Para realizar uma checagem de alertas, basta invocar a função *checar()*, passando como parâmetro uma instância de um cliente SAT (*ClienteSATLocal* ou *ClienteSATHub*):

```
sat = ClienteSATHub('10.0.0.200', port=5000)
alertas = checar(sat)
if alertas:
    # existem alertas ativos...
    faz_algo_a_respeito()
```

Este módulo fornece os seguintes alertas:

- **Documentos Pendentes** Detecta a existência de um ou mais CF-e-SAT pendentes de transmissão para a SEFAZ. Fornecido pela classe *AlertaCFePendentes*.
- **Vencimento do Certificado** Detecta se o vencimento do certificado instalado no equipamento SAT está se aproximando (ou se já venceu). Fornecido pela classe *AlertaVencimentoCertificado*.
- **Divergência de Horários** Detecta se a diferença entre o horário do sistema e do equipamento SAT é superior ao tolerado. Fornecido pela classe *AlertaDivergenciaHorarios*.

É fácil implementar outros alertas se necessário, bastando implementar uma subclasse de *AlertaOperacao* e registrando a classe através da função *registrar()*:

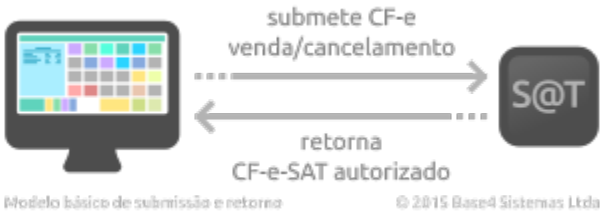
```
from satcfe.alertas import AlertaOperacao
from satcfe.alertas import registrar

class OutroAlerta(AlertaOperacao):

    def checar(self):
        if condicao:
            self._ativo = True
        return self._ativo

    def mensagem(self):
        if self._ativo:
            return 'Este alerta está ativo por uma razão.'
        return 'Este alerta não está ativo.'

registrar(OutroAlerta)
```

Se você desenvolver algum alerta, considere compartilhar a sua implementação. Caso você note que algum equipamento tenha resultado informações inesperadas, fazendo com que algum alerta não funcione conforme o esperado, avise-nos, preenchendo um relatório do problema.

Veja também a documentação da *API da Infraestrutura de Alertas*.

3.6 Exemplos de Documentos

Abaixo estão relacionados documentos CF-e que formam um ciclo completo, desde o XML de venda gerado pelo aplicativo comercial e seu retorno autorizado pelo equipamento SAT e o seu subsequente cancelamento.



3.6.1 XML do CF-e de Venda

O seguinte documento XML representa um CF-e de venda pronto para ser enviado ao equipamento SAT. Um documento como este pode ser criado como visto em *Criando um CF-e de Venda* e submetido às funções SAT `enviar_dados_venda()` e/ou `teste_fim_a_fim()`.

```
<?xml version="1.0"?>
<CFe>
  <infCFe versaoDadosEnt="0.07">
    <ide>
      <CNPJ>08427847000169</CNPJ>
      <signAC>SGR-SAT SISTEMA DE GESTAO E RETAGUARDA DO SAT</signAC>
      <numeroCaixa>001</numeroCaixa>
    </ide>
    <emit>
```

(continues on next page)

(continuação da página anterior)

```
      <CNPJ>61099008000141</CNPJ>
      <IE>111111111111</IE>
      <IM>12345</IM>
      <cRegTribISSQN>3</cRegTribISSQN>
      <indRatISSQN>N</indRatISSQN>
    </emit>
    <dest/>
    <det nItem="1">
      <prod>
        <cProd>116</cProd>
        <cEAN>9990000001163</cEAN>
        <xProd>Cascao</xProd>
        <CFOP>5405</CFOP>
        <uCom>UN</uCom>
        <qCom>1.0000</qCom>
        <vUnCom>4.00</vUnCom>
        <indRegra>A</indRegra>
      </prod>
      <imposto>
        <ICMS>
          <ICMSSN102>
            <Orig>0</Orig>
            <CSOSN>500</CSOSN>
          </ICMSSN102>
        </ICMS>
        <PIS>
          <PISSN>
            <CST>49</CST>
          </PISSN>
        </PIS>
        <COFINS>
          <COFINSSN>
            <CST>49</CST>
          </COFINSSN>
        </COFINS>
      </imposto>
    </det>
    <total/>
    <pgto>
      <MP>
        <cMP>01</cMP>
        <vMP>4.00</vMP>
      </MP>
    </pgto>
  </infCFe>
</CFe>
```

3.6.2 XML do CF-e-SAT de Venda

O seguinte documento XML **seria um documento fiscal com validade jurídica** se não tivesse sido **emitido contra um equipamento SAT para desenvolvimento**[1]. Repare que o emitente possui os dados do fabricante do equipamento além de vários outros elementos importantes que foram adicionados pelo equipamento, tais como o valor do troco e o bloco de assinatura no final do documento.

[1] Também são chamados de "kit SAT".

```
<?xml version="1.0"?>
<CFe>
  <infCFe Id="CFe35161261099008000141599000026310003024947916" versao="0.07"␣
↪versaoDadosEnt="0.07" versaoSB="010300">
    <ide>
      <cUF>35</cUF>
      <cNF>494791</cNF>
      <mod>59</mod>
      <nserieSAT>900002631</nserieSAT>
      <nCFe>000302</nCFe>
      <dEmi>20161220</dEmi>
      <hEmi>095111</hEmi>
      <cDV>6</cDV>
      <tpAmb>2</tpAmb>
      <CNPJ>08427847000169</CNPJ>
      <signAC>SGR-SAT SISTEMA DE GESTAO E RETAGUARDA DO SAT</signAC>
      <assinaturaQRCODE>
↪EKP6q5FOMSxPfTr4yf2LHBw4UTp6vsFsJ6cM3c6Lc0AjLExZ63tERLucVgp5Ao69fzmS103/
↪PiTyw2XdweVebq1hfLiK7vbPRqgVWJySxjcLUCzMJacVlPCJyOwTxDL34tyvbW6Vr+c2+jBB3HsqO3zxW9ZsrzBWhhkxx9zWgF
↪1+WZz1ESoU+kGfpVu+z45j70AZu0bIRjFT6bqs65BUDtUKRtMsq72vocnSD9yQgYHwNpZglCGkREyF2qQu18oqN0RQvi9L8DBS
↪</assinaturaQRCODE>
      <numeroCaixa>001</numeroCaixa>
    </ide>
    <emit>
      <CNPJ>61099008000141</CNPJ>
      <xNome>DIMAS DE MELO PIMENTA SISTEMAS DE PONTO E ACESSO LTDA</xNome>
      <xFant>DIMEP</xFant>
      <enderEmit>
        <xLgr>AVENIDA MOFARREJ</xLgr>
        <nro>840</nro>
        <xCpl>908</xCpl>
        <xBairro>VL. LEOPOLDINA</xBairro>
        <xMun>SAO PAULO</xMun>
        <CEP>05311000</CEP>
      </enderEmit>
      <IE>111111111111</IE>
      <IM>12345</IM>
      <cRegTrib>3</cRegTrib>
      <cRegTribISSQN>3</cRegTribISSQN>
      <indRatISSQN>N</indRatISSQN>
    </emit>
    <dest/>
    <det nItem="1">
      <prod>
        <cProd>116</cProd>
        <cEAN>9990000001163</cEAN>
        <xProd>Cascao</xProd>
        <CFOP>5405</CFOP>
        <uCom>UN</uCom>
        <qCom>1.0000</qCom>
        <vUnCom>4.00</vUnCom>
        <vProd>4.00</vProd>
        <indRegra>A</indRegra>
        <vItem>4.00</vItem>
      </prod>
      <imposto>
        <ICMS>
```

(continues on next page)

(continuação da página anterior)

```
          <ICMSSN102>
            <Orig>0</Orig>
            <CSOSN>500</CSOSN>
          </ICMSSN102>
        </ICMS>
        <PIS>
          <PISSN>
            <CST>49</CST>
          </PISSN>
        </PIS>
        <COFINS>
          <COFINSSN>
            <CST>49</CST>
          </COFINSSN>
        </COFINS>
      </imposto>
    </det>
    <total>
      <ICMSTot>
        <vICMS>0.00</vICMS>
        <vProd>4.00</vProd>
        <vDesc>0.00</vDesc>
        <vPIS>0.00</vPIS>
        <vCOFINS>0.00</vCOFINS>
        <vPISST>0.00</vPISST>
        <vCOFINSST>0.00</vCOFINSST>
        <vOutro>0.00</vOutro>
      </ICMSTot>
      <vCFe>4.00</vCFe>
    </total>
    <pgto>
      <MP>
        <cMP>01</cMP>
        <vMP>4.00</vMP>
      </MP>
      <vTroco>0.00</vTroco>
    </pgto>
  </infCFe>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
↪20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
      <Reference URI="#CFe35161261099008000141599000026310003024947916">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature
↪"/>
          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
        <DigestValue>3sW0ay6BB4wbJUVnp9B1SQpqCka+ga8gwxHy/ViCXuw=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
↪G9hWhCmhaFn4FYNU+ukYm4a8OfvFR5fZz9CYGj0XN5ayr43p2ugho7oCY1ySlwhrl0dTfnvQrE6S1IvD/
↪W7LcS8PSnCx4G7X4wzZpTQojDAwbAZBBHjEe4Xhj7gIHRKhHvwiHwBsuBF64zZgUCcs91SgVJNFc1pfor/
↪RP37pfwfFNQtobhvvFak99J0aPsHsmYoPeQBH2HzrmGfqLvZvpZQX7xJlpgU/D/
↪wPxWpSTjHUOr2UegT0LYMPudpUQ96rBLgByvwx2rkJ+fmXz94T9WNPQxRyhJduDjChCFkwqP2+9Ik4+S3LPQDVM/
↪MonHvXB9BPIYRJOjiN3C6Nc3WMSA==</SignatureValue>
```

(continues on next page)

(continuação da página anterior)

```
    <KeyInfo>
      <X509Data>
        <X509Certificate>
↪MIIG7zCCBNegAwIBAgIQPYgMyeNxYCOCHVsMXFBvODANBgkqhkiG9w0BAQsFADBnMQswCQYDVQQGEwJCUjE1MDMGA1UEChMsU2V
↪Sg4cOokkV7WYxRjxFLNJEnFb3n31kmhlqICAAYRTazeJuaR/qkuvc5MjyVmI5cMR+GWhvrOK7Dm4y8kpMyJ/
↪Kqo8A887jUlqpUs4aJ+TwUnK0w8Hf7SUYofwteKPlfXsEHbpn3kJHVoUyNMnIu8nkqdlhnYXWwPBbn56Xc2aZgJS4IQFd/
↪z/
↪C4T01KC5f31jehZTc+ColHsvG6xgH9dEx9Bk9NVaPBMYBuNNOJOEOWw+Lh1cc8Jn1UOOOTDyiOy8vfzCVIeMRVncY1mKFtHy4Dr
↪BAQDAgXgMB8GA1UdIwQYMBaAFI45QQBc8rgF2qhtmLkBRm1uY98CMGsGA1UdHwRkMGIwYKBeoFyGWmh0dHA6Ly9hY3NhdC10ZXN
↪1Mjqi0+SV82g2mf7gbKNvEV9w5gKrTGw6rkTTYf5HpqPtb3KNxsCeKpAfkdupT8WkRM9FANfW0kPH2adHdcNOdEfEiSmOIjFVnT
↪eTVbzflwqhBpAXzyYWN2bCZDqlFNAhib1vIe/
↪cz8i6OHYrXk602qw4vnAcOpB0rlHtZCXIUiCZCanBjdn5PmSZVh88bzpJd3ltMd116YFAyShSJXCi8SqOLRVzQXkXvbL0iUqg6T
↪mCvTQCuuDq4EGljXW1FAy8rS732r4BKzJ7xWo5BGZKZp4jANo62cECSJhApwzQBnfiDWil353rtxGUweTP92dJGcGraiLHP7wuI
↪hRAUiDpliiQZaBaXsKxVk4uxFKn7/
↪86BB4GTqTQMW4XXzE6hG4PhwriPiG9cPYDt68hR2LK1vHZXzBc6P3QxGlh/
↪rdiJMpzt6RY5luEciP1+LI8YCZVIvqY0YZwoCG3vVkDYwpNpHnlZVct2ugYBCd9cDgXRUD3kO0GU2P+xnaiAMfsLSo3JhfXzi5i
↪</X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
</CFe>
```

3.6.3 XML do CF-e de Cancelamento

O seguinte documento XML representa um CF-e de cancelamento pronto para ser enviado ao equipamento SAT. Um documento como este pode ser criado como visto em *Criando um CF-e de Cancelamento* e submetido à função SAT *cancelar_ultima_venda()*.

```
<?xml version="1.0"?>
<CFeCanc>
  <infCFe chCanc="CFe35161261099008000141599000026310003024947916">
    <ide>
      <CNPJ>08427847000169</CNPJ>
      <signAC>SGR-SAT SISTEMA DE GESTAO E RETAGUARDA DO SAT</signAC>
      <numeroCaixa>001</numeroCaixa>
    </ide>
    <emit/>
    <dest/>
    <total/>
  </infCFe>
</CFeCanc>
```

3.6.4 XML do CF-e-SAT de Cancelamento

O seguinte documento XML **seria um documento fiscal com validade jurídica** se não tivesse sido **emitido contra um equipamento SAT para desenvolvimento**. Repare que o equipamento SAT adiciona vários outros elementos ao documento antes de assiná-lo e enviá-lo à SEFAZ.

```
<?xml version="1.0"?>
<CFeCanc>
  <infCFe Id="CFe35161261099008000141599000026310003038725260" chCanc=
↪"CFe35161261099008000141599000026310003024947916" versao="0.07">
    <dEmi>20161220</dEmi>
```

(continues on next page)

(continuação da página anterior)

```
    <hEmi>095111</hEmi>
    <ide>
      <cUF>35</cUF>
      <cNF>872526</cNF>
      <mod>59</mod>
      <nserieSAT>900002631</nserieSAT>
      <nCFe>000303</nCFe>
      <dEmi>20161220</dEmi>
      <hEmi>095203</hEmi>
      <cDV>0</cDV>
      <CNPJ>08427847000169</CNPJ>
      <signAC>SGR-SAT SISTEMA DE GESTAO E RETAGUARDA DO SAT</signAC>
      <assinaturaQRCODE>XbWIYJSkzTJyixs9ZHOn07dUFuxFFZYAd/slcOj2WjTUqwDHVAYk5v2efko6/
↪0tE3rtEJ3Zm1vAyoXeG12i+OYJaykoGzUdyTtSkwZdT1FnPoOkHzPipR+Be7GmaxYBkWD/
↪ytoGd6bApFvlWbxOGqmW1Ngo4mIr4Rn14TL31tGXz4HAuTCnIP/bKkQlJ8R1d1+4SB06DbEM/
↪8QC3mFLzKTH2T7rEtqF7KTLdIrovzyeIAL6ci6AebB/
↪wpna9KcfWiC2zoAc0rsp08zQUe5B+8kE6KPjPWvQkSOndhIdxLOt45v7rsl72J0qjIVzSSCkj3UtCitAmorkrfCtKTWi/
↪WA==</assinaturaQRCODE>
      <numeroCaixa>001</numeroCaixa>
    </ide>
    <emit>
      <CNPJ>61099008000141</CNPJ>
      <xNome>DIMAS DE MELO PIMENTA SISTEMAS DE PONTO E ACESSO LTDA</xNome>
      <xFant>DIMEP</xFant>
      <enderEmit>
        <xLgr>AVENIDA MOFARREJ</xLgr>
        <nro>840</nro>
        <xCpl>908</xCpl>
        <xBairro>VL. LEOPOLDINA</xBairro>
        <xMun>SAO PAULO</xMun>
        <CEP>05311000</CEP>
      </enderEmit>
      <IE>111111111111</IE>
      <IM>12345</IM>
    </emit>
    <dest/>
    <total>
      <vCFe>4.00</vCFe>
    </total>
  </infCFe>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
↪20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
      <Reference URI="#CFe35161261099008000141599000026310003038725260">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature
↪"/>
          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
        <DigestValue>tzvG+236ZjAEMmwraPQoLGEBXNYJ3YAWoJ+5C4NXzdc=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
↪ciPsCbdN8RHYJ13BqkYY+IiR3DePjCfehK8XBukyMvCsBVkNrM++nQyIgTWeAXhYuCWFsoX3BH1GLAvHQPG1xqnynqh2UFQmgxu
↪Q4HF9OYzA6frylWMSfmZzqM64DHmXhhDr2YAggFneRDdBZy9qOYAewZux96vbeqHNrc0SHTmmWwDjrrDkGre9aOxgT8u1tMVQIJ
↪fHT8KK+S84lM02md/woEld4hI5B7lRb9mCWWR5T5OIx7JYqSkQtMc+vNRN+N/
↪XkuWtq21EWRTnhqk3oAewOg+gBw==</SignatureValue>
```

(continues on next page)

(continuação da página anterior)

```
    <KeyInfo>
      <X509Data>
        <X509Certificate>
↪MIIG7zCCBNegAwIBAgIQPYgMyeNxYCOCHVsMXFBvODANBgkqhkiG9w0BAQsFADBnMQswCQYDVQQGEwJCUjE1MDMGA1UEChMsU2V
↪Sg4cOokkV7WYxRjxFLNJEnFb3n31kmhlqICAAYRTazeJuaR/qkuvc5MjyVmI5cMR+GWhvrOK7Dm4y8kpMyJ/
↪Kqo8A887jUlqpUs4aJ+TwUnK0w8Hf7SUYofwteKPlfXsEHbpn3kJHVoUyNMnIu8nkqdlhnYXWwPBbn56Xc2aZgJS4IQFd/
↪z/
↪C4T01KC5f31jehZTc+ColHsvG6xgH9dEx9Bk9NVaPBMYBuNNOJOEOWw+Lh1cc8Jn1UOOOTDyiOy8vfzCVIeMRVncY1mKFtHy4Dr
↪BAQDAgXgMB8GA1UdIwQYMBaAFI45QQBc8rgF2qhtmLkBRm1uY98CMGsGA1UdHwRkMGIwYKBeoFyGWmh0dHA6Ly9hY3NhdC10ZXN
↪1Mjqi0+SV82g2mf7gbKNvEV9w5gKrTGw6rkTTYf5HpqPtb3KNxsCeKpAfkdupT8WkRM9FANfW0kPH2adHdcNOdEfEiSmOIjFVnT
↪eTVbzflwqhBpAXzyYWN2bCZDqlFNAhib1vIe/
↪cz8i6OHYrXk602qw4vnAcOpB0rlHtZCXIUiCZCanBjdn5PmSZVh88bzpJd3ltMd116YFAyShSJXCi8SqOLRVzQXkXvbL0iUqg6T
↪mCvTQCuuDq4EGljXW1FAy8rS732r4BKzJ7xWo5BGZKZp4jANo62cECSJhApwzQBnfiDWil353rtxGUweTP92dJGcGraiLHP7wuI
↪hRAUiDpliiQZaBaXsKxVk4uxFKn7/
↪86BB4GTqTQMW4XXzE6hG4PhwriPiG9cPYDt68hR2LK1vHZXzBc6P3QxGlh/
↪rdiJMpzt6RY5luEciP1+LI8YCZVIvqY0YZwoCG3vVkDYwpNpHnlZVct2ugYBCd9cDgXRUD3kO0GU2P+xnaiAMfsLSo3JhfXzi5i
↪</X509Certificate>
      </X509Data>
    </KeyInfo>
  </Signature>
</CFeCanc>
```

Notas

3.7 Documentação da API

Os módulos *satcfe.base*, *satcfe.clientelocal* e *satcfe.clientesathub* são a fundação para comunicação com o equipamento SAT conectado à máquina local ou à um equipamento SAT compartilhado através de um servidor SATHub.

3.7.1 Módulo satcfe.base

class satcfe.base.**BibliotecaSAT**(*caminho*, *convencao=None*)

Configura a localização da biblioteca que efetivamente acessará o equipamento SAT. A biblioteca deverá ser uma DLL (*dynamic linked library*, em sistemas Microsoft Windows) ou uma *shared library* em sistemas baseados no UNIX ou GNU/Linux.

Parâmetros

- **caminho** (*string*) – Caminho completo para a biblioteca SAT.
- **convencao** (*integer*) – Opcional. Indica a convenção de chamada da biblioteca, devendo ser uma das constantes definidas em CONVENCOES_CHAMADA. Se não for informado, a convenção de chamada será decidida conforme a extensão do nome do arquivo, assumindo WINDOWS_STDCALL para as extensões .DLL ou .dll. Quaisquer outras extensões, assume a convenção de chamada STANDARD_C.

caminho

Caminho completo para a biblioteca SAT.

convencao

Convenção de chamada para a biblioteca SAT. Deverá ser um dos valores disponíveis na contante CONVENCOES_CHAMADA.

ref
Uma referência para a biblioteca SAT carregada.

class satcfe.base.**FuncoesSAT**(*biblioteca*, *codigo_ativacao=None*, *numerador_sessao=None*, *encoding='utf-8'*, *encoding_errors='strict'*)

Estabelece a interface básica para acesso às funções da biblioteca SAT.

A intenção é que esta classe seja a base para classes mais especializadas capazes de trabalhar as respostas, resultando em objetos mais úteis, já que os métodos desta classe invocam as funções da biblioteca SAT e retornam o resultado *verbatim*.

As funções implementadas estão descritas na ER SAT, item 6.1.

Item ER	Função	Método
6.1.1	AtivarSAT	*ativar_sat()*
6.1.2	ComunicarCertificadoICPBRASIL	*comunicar_certificado_icpbrasil()*
6.1.3	EnviarDadosVenda	*enviar_dados_venda()*
6.1.4	CancelarUltimaVenda	*cancelar_ultima_venda()*
6.1.5	ConsultarSAT	*consultar_sat()*
6.1.6	TesteFimAFim	*teste_fim_a_fim()*
6.1.7	ConsultarStatusOperacional	*consultar_status_operacional()*
6.1.8	ConsultarNumeroSessao	*consultar_numero_sessao()*
6.1.9	ConfigurarInterfaceDeRede	*configurar_interface_de_rede()*
6.1.10	AssociarAssinatura	*associar_assinatura()*
6.1.11	AtualizarSoftwareSAT	*atualizar_software_sat()*
6.1.12	ExtrairLogs	*extrair_logs()*
6.1.13	BloquearSAT	*bloquear_sat()*
6.1.14	DesbloquearSAT	*desbloquear_sat()*
6.1.15	TrocarCodigoDeAtivacao	*trocar_codigo_de_ativacao()*
6.1.16	ConsultarUltimaSessaoFiscal	*consultar_ultima_sessao_fiscal()*

Parâmetros

- **biblioteca** – Uma instância de *BibliotecaSAT*.
- **codigo_ativacao** (*string*) – Código de ativação. Senha definida pelo contribuinte no software de ativação, conforme item 2.1.1 da ER SAT.
- **numerador_sessao** – Opcional. Um callable capaz de gerar um número de sessão conforme descrito no item 6, alínea "a", "Funções do Equipamento SAT", da ER SAT. Se não for especificado, será utilizado um *NumeroSessaoMemoria*.

associar_assinatura(*sequencia_cnpj*, *assinatura_ac*)

Função AssociarAssinatura conforme ER SAT, item 6.1.10. Associação da assinatura do aplicativo comercial.

Parâmetros

- **sequencia_cnpj** – Sequência string de 28 dígitos composta do CNPJ do desenvolvedor da AC e do CNPJ do estabelecimento comercial contribuinte, conforme ER SAT, item 2.1.3.
- **assinatura_ac** – Sequência string contendo a assinatura digital do parâmetro sequencia_cnpj codificada em base64.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

ativar_sat(*tipo_certificado*, *cnpj*, *codigo_uf*)

Função `AtivarSAT` conforme ER SAT, item 6.1.1. Ativação do equipamento SAT. Dependendo do tipo do certificado, o procedimento de ativação é complementado enviando-se o certificado emitido pela ICP-Brasil (*comunicar_certificado_icpbrasil()*).

Parâmetros

- **tipo_certificado** (*int*) – Deverá ser um dos valores `satcomum.constantes.CERTIFICADO_ACSAT_SEFAZ`, `satcomum.constantes.CERTIFICADO_ICPBRASIL` ou `satcomum.constantes.CERTIFICADO_ICPBRASIL_RENOVACAO`, mas nenhuma validação será realizada antes que a função de ativação seja efetivamente invocada.
- **cnpj** (*str*) – Número do CNPJ do estabelecimento contribuinte, contendo apenas os dígitos. Nenhuma validação do número do CNPJ será realizada antes que a função de ativação seja efetivamente invocada.
- **codigo_uf** (*int*) – Código da unidade federativa onde o equipamento SAT será ativado (eg. `35` para o Estado de São Paulo). Nenhuma validação do código da UF será realizada antes que a função de ativação seja efetivamente invocada.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

atualizar_software_sat()

Função `AtualizarSoftwareSAT` conforme ER SAT, item 6.1.11, Atualização do software do equipamento SAT.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

bloquear_sat()

Função `BloquearSAT` conforme ER SAT, item 6.1.13. Bloqueio operacional do equipamento SAT.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

cancelar_ultima_venda(*chave_cfe*, *dados_cancelamento*, **args*, ***kwargs*)

Função `CancelarUltimaVenda` conforme ER SAT, item 6.1.4. Envia o CF-e de cancelamento para o equipamento SAT, que o enviará para autorização e cancelamento do CF-e pela SEFAZ.

Parâmetros

- **chave_cfe** – String contendo a chave do CF-e de venda que será cancelado (deve possuir o prefixo `"CFe"` seguido dos dígitos da chave do CF-e-SAT autorizado anteriormente).
- **dados_cancelamento** – Uma instância de *CFeCancelamento* ou uma string contendo o XML do CF-e de cancelamento.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

comunicar_certificado_icpbrasil(*certificado*)

Função `ComunicarCertificadoICPBRASIL` conforme ER SAT, item 6.1.2. Envio do certificado criado pela ICP-Brasil.

Parâmetros **certificado** (*str*) – Conteúdo do certificado digital criado pela autoridade certificadora ICP-Brasil.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

configurar_interface_de_rede(*configuracao*, **args*, ***kwargs*)
Função ConfigurarInterfaceDeRede conforme ER SAT, item 6.1.9. Configuração da interface de comunicação do equipamento SAT.

Parâmetros **configuracao** – Um objeto *ConfiguracaoRede* ou uma string contendo o XML com as configurações de rede.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

consultar_numero_sessao(*numero_sessao*)
Função ConsultarNumeroSessao conforme ER SAT, item 6.1.8. Consulta o equipamento SAT por um número de sessão específico.

Parâmetros **numero_sessao** (*int*) – Número da sessão que se quer consultar.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

consultar_sat()
Função ConsultarSAT conforme ER SAT, item 6.1.5. Usada para testes de comunicação entre a AC e o equipamento SAT.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno str

consultar_status_operacional()
Função ConsultarStatusOperacional conforme ER SAT, item 6.1.7. Consulta do status operacional do equipamento SAT.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

consultar_ultima_sessao_fiscal()
Função ConsultarUltimaSessaoFiscal conforme ER SAT, item 6.1.16. Retorna a resposta da última sessão fiscal, isto é, do último comando fiscal (EnviarDadosVenda ou CancelarUltimaVenda) executado pelo equipamento SAT.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

desbloquear_sat()
Função DesbloquearSAT conforme ER SAT, item 6.1.14. Desbloqueio operacional do equipamento SAT.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

enviar_dados_venda(*dados_venda*, **args*, ***kwargs*)
Função EnviarDadosVenda conforme ER SAT, item 6.1.3. Envia o CF-e de venda para o equipamento SAT, que o enviará para autorização pela SEFAZ.

Parâmetros **dados_venda** – Instância de *CFeVenda* ou uma string contendo o XML do CF-e de venda.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

extrair_logs()
Função ExtrairLogs conforme ER SAT, item 6.1.12. Extração dos registros de log do equipamento SAT.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

gerar_numero_sessao()
Gera o número de sessão para a próxima invocação de função SAT.

teste_fim_a_fim(*dados_venda*, **args*, ***kwargs*)
Função TesteFimAFim conforme ER SAT, item 6.1.6. Teste de comunicação entre a AC, o equipamento SAT e a SEFAZ.

Parâmetros **dados_venda** – Instância de *CFeVenda* ou uma string contendo o XML do CF-e de venda de teste.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

trocar_codigo_de_ativacao(*novo_codigo_ativacao*, *opcao=1*, *codigo_emergencia=None*)
Função TrocarCodigoDeAtivacao conforme ER SAT, item 6.1.15. Troca do código de ativação do equipamento SAT. YouX.

Parâmetros

- **novo_codigo_ativacao** (*str*) – O novo código de ativação escolhido pelo contribuinte.
- **opcao** (*int*) – Indica se deverá ser utilizado o código de ativação atualmente configurado, que é um código de ativação regular, definido pelo contribuinte, ou se deverá ser usado um código de emergência. Deverá ser o valor de uma das constantes satcomum.constantes.CODIGO_ATIVACAO_REGULAR (padrão) ou satcomum.constantes.CODIGO_ATIVACAO_EMERGENCIA. Nenhuma validação será realizada antes que a função seja efetivamente invocada. Entretanto, se opção de código de ativação indicada for CODIGO_ATIVACAO_EMERGENCIA, então o argumento que informa o codigo_emergencia será checado e deverá avaliar como verdadeiro.
- **codigo_emergencia** (*str*) – O código de ativação de emergência, que é definido pelo fabricante do equipamento SAT. Este código deverá ser usado quando o usuário perder o código de ativação regular, e precisar definir um novo código de ativação. Note que, o argumento opcao deverá ser informado com o valor satcomum.constantes.CODIGO_ATIVACAO_EMERGENCIA para que este código de emergência seja considerado.

Retorna Retorna *verbatim* a resposta da função SAT.

Tipo de retorno string

Levanta **ValueError** – Se o novo código de ativação avaliar como falso (possuir uma string nula por exemplo) ou se o código de emergencia avaliar como falso quando a opção for pelo código de ativação de emergência.

Aviso: De acordo com a *ER SAT*, a função TrocarCodigoDeAtivacao requer que o novo código de ativação seja especificado duas vezes.

> Este método ignora isso e apenas informa o mesmo código de ativação duas vezes na chamada da função. O entendimento é de que a confirmação do código de ativação é responsabilidade de outras camadas da aplicação (eg. interface com o usuário) e, portanto, fora do escopo desta biblioteca.

class satcfe.base.**NumeroSessaoMemoria**(*tamanho=100*)

Implementa um numerador de sessão simples, baseado em memória, não persistente, que irá gerar um número de sessão (seis dígitos) diferente entre os `n` últimos números de sessão gerados. Conforme a ER SAT, um número de sessão não poderá ser igual aos últimos `100` números.

3.7.2 Módulo satcfe.clientelocal

class satcfe.clientelocal.**ClienteSATLocal**(**args*, ***kwargs*)

Fornece acesso ao equipamento SAT conectado na máquina local.

As respostas às funções SAT serão trabalhadas resultando em objetos Python regulares cujos atributos representam as peças de informação conforme descrito, função por função, na ER SAT.

associar_assinatura(*sequencia_cnpj*, *assinatura_ac*)

Sobrepõe *associar_assinatura()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

ativar_sat(*tipo_certificado*, *cnpj*, *codigo_uf*)

Sobrepõe *ativar_sat()*.

Retorna Uma resposta SAT especilizada em `AtivarSAT`.

Tipo de retorno *satcfe.resposta.ativarsat.RespostaAtivarSAT*

atualizar_software_sat()

Sobrepõe *atualizar_software_sat()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

bloquear_sat()

Sobrepõe *bloquear_sat()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

cancelar_ultima_venda(*chave_cfe*, *dados_cancelamento*, **args*, ***kwargs*)

Sobrepõe *cancelar_ultima_venda()*.

Retorna Uma resposta SAT especializada em `CancelarUltimaVenda`.

Tipo de retorno *satcfe.resposta.cancelarultimavenda.RespostaCancelarUltimaVenda*

comunicar_certificado_icpbrasil(*certificado*)

Sobrepõe *comunicar_certificado_icpbrasil()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

configurar_interface_de_rede(*configuracao*, **args*, ***kwargs*)

Sobrepõe *configurar_interface_de_rede()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

consultar_numero_sessao(*numero_sessao*)
Sobrepõe *consultar_numero_sessao()*.

Retorna Uma resposta SAT que irá depender da sessão consultada.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

consultar_sat()
Sobrepõe *consultar_sat()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

consultar_status_operacional()
Sobrepõe *consultar_status_operacional()*.

Retorna Uma resposta SAT especializada em `ConsultarStatusOperacional`.

Tipo de retorno *satcfe.resposta.consultarstatusoperacional.RespostaConsultarStatusOperacional*

consultar_ultima_sessao_fiscal()
Sobrepõe *consultar_ultima_sessao_fiscal()*.

Retorna Uma resposta SAT que irá depender do último comando "fiscal" executado pelo equipmamento SAT, que poderá ser uma venda ou um cancelamento de venda.

Tipo de retorno satcfe.resposta.consultarultimasessaofiscal.RespostaConsultarUltimaSessaoFiscal | satcfe.resposta.enviardadosvenda.RespostaEnviarDadosVenda | satcfe.resposta.cancelarultimavenda.RespostaCancelarUltimaVenda

desbloquear_sat()
Sobrepõe *desbloquear_sat()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

enviar_dados_venda(*dados_venda*, **args*, ***kwargs*)
Sobrepõe *enviar_dados_venda()*.

Retorna Uma resposta SAT especializada em `EnviarDadosVenda`.

Tipo de retorno *satcfe.resposta.enviardadosvenda.RespostaEnviarDadosVenda*

extrair_logs()
Sobrepõe *extrair_logs()*.

Retorna Uma resposta SAT especializada em `ExtrairLogs`.

Tipo de retorno *satcfe.resposta.extrairlogs.RespostaExtrairLogs*

teste_fim_a_fim(*dados_venda*, **args*, ***kwargs*)
Sobrepõe *teste_fim_a_fim()*.

Retorna Uma resposta SAT especializada em `TesteFimAFim`.

Tipo de retorno *satcfe.resposta.testefimafim.RespostaTesteFimAFim*

trocar_codigo_de_ativacao(*novo_codigo_ativacao*, *opcao=1*, *codigo_emergencia=None*)
Sobrepõe *trocar_codigo_de_ativacao()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

3.7.3 Módulo `satcfe.clientesathub`

class satcfe.clientesathub.**ClienteSATHub**(*host*, *port*, *numero_caixa=1*, *baseurl='/hub/v1'*)

Fornece acesso concorrente a um equipamento SAT remoto.

O acesso é feito consumindo-se a API RESTful SATHub que irá efetivamente acessar um equipamento SAT e responder através de uma conexão HTTP.

As respostas às funções SAT serão trabalhadas resultando em objetos Python regulares cujos atributos representam as peças de informação conforme descrito, função por função, na ER SAT.

Parâmetros

- **host** (*string*) – Nome ou endereço IP do host para o SATHub.
- **port** (*integer*) – Número da porta pela qual o HTTPd responde.
- **numero_caixa** (*integer*) – Número do caixa, conforme atributo `B14` do item 4.2.2 da ER SAT. Deve ser um número inteiro entre `0` e `999`. Na verdade, prefira deixar o número de caixa `999` livre, para uso pelo próprio SATHub.
- **baseurl** (*string*) – Opcional. Prefixo base da URL para os serviços da API RESTful. Se não for informado será utilizado o padrão `"/hub/v1"`.

Nota: Note que não é necessário especificar o código de ativação quando se está usando um *ClienteSATHub*, já que o código é configurado no servidor.

---

associar_assinatura(*sequencia_cnpj*, *assinatura_ac*)

Sobrepõe *associar_assinatura()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

ativar_sat(*tipo_certificado*, *cnpj*, *codigo_uf*)

Sobrepõe *ativar_sat()*.

Retorna Uma resposta SAT especializada em `AtivarSAT`.

Tipo de retorno *satcfe.resposta.ativarsat.RespostaAtivarSAT*

atualizar_software_sat()

Sobrepõe *atualizar_software_sat()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

bloquear_sat()

Sobrepõe *bloquear_sat()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

cancelar_ultima_venda(*chave_cfe*, *dados_cancelamento*, **args*, ***kwargs*)

Sobrepõe *cancelar_ultima_venda()*.

Retorna Uma resposta SAT especializada em `CancelarUltimaVenda`.

Tipo de retorno *satcfe.resposta.cancelarultimavenda.RespostaCancelarUltimaVenda*

comunicar_certificado_icpbrasil(*certificado*)
Sobrepõe *comunicar_certificado_icpbrasil()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

configurar_interface_de_rede(*configuracao*, **args*, ***kwargs*)
Sobrepõe *configurar_interface_de_rede()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

consultar_numero_sessao(*numero_sessao*)
Sobrepõe *consultar_numero_sessao()*.

Retorna Uma resposta SAT que irá depender da sessão consultada.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

consultar_sat()
Sobrepõe *consultar_sat()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

consultar_status_operacional()
Sobrepõe *consultar_status_operacional()*.

Retorna Uma resposta SAT especializada em `ConsultarStatusOperacional`.

Tipo de retorno *satcfe.resposta.consultarstatusoperacional.RespostaConsultarStatusOperacional*

consultar_ultima_sessao_fiscal()
Sobrepõe *consultar_ultima_sessao_fiscal()*.

Retorna Uma resposta SAT que irá depender do último comando "fiscal" executado pelo equipmamento SAT, que poderá ser uma venda ou um cancelamento de venda.

Tipo de retorno satcfe.resposta.consultarultimasessaofiscal.RespostaConsultarUltimaSessaoFiscal | satcfe.resposta.enviardadosvenda.RespostaEnviarDadosVenda | satcfe.resposta.cancelarultimavenda.RespostaCancelarUltimaVenda

desbloquear_sat()
Sobrepõe *desbloquear_sat()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

enviar_dados_venda(*dados_venda*, **args*, ***kwargs*)
Sobrepõe *enviar_dados_venda()*.

Retorna Uma resposta SAT especializada em `EnviarDadosVenda`.

Tipo de retorno *satcfe.resposta.enviardadosvenda.RespostaEnviarDadosVenda*

extrair_logs()
Sobrepõe *extrair_logs()*.

Retorna Uma resposta SAT especializada em `ExtrairLogs`.

Tipo de retorno *satcfe.resposta.extrairlogs.RespostaExtrairLogs*

teste_fim_a_fim(*dados_venda*, **args*, ***kwargs*)
Sobrepõe *teste_fim_a_fim()*.

Retorna Uma resposta SAT especializada em TesteFimAFim.

Tipo de retorno *satcfe.resposta.testefimafim.RespostaTesteFimAFim*

trocar_codigo_de_ativacao(*novo_codigo_ativacao*, *opcao=2*, *codigo_emergencia=None*)
Sobrepõe *trocar_codigo_de_ativacao()*.

Retorna Uma resposta SAT padrão.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

3.7.4 Módulo satcfe.entidades

class satcfe.entidades.**CFeCancelamento**(*destinatario=None*, ***kwargs*)
Representa um CF-e de cancelamento.

Parâmetros

- **destinatario** (Destinatario) – Opcional. Uma instância de *Destinatario* contendo apenas os dados exigidos para a operação de cancelamento (ie. CPF ou CNPJ do destinatário).
- **chCanc** (*str*) – Chave de acesso do CF-e a ser cancelado. Deve ser precedido do literal CFe seguido dos quarenta e quatro dígitos que compõem a chave de acesso.
- **CNPJ** (*str*) – CNPJ da software house, desenvolvedora do aplicativo comercial, contendo apenas os dígitos do número e incluindo zeros não significativos, se for o caso (14 dígitos).
- **signAC** (*str*) – Assinatura do aplicativo comercial (344 dígitos).
- **numeroCaixa** (*int*) – Número do caixa ao qual o SAT está conectado. Normalmente este será o número do caixa de onde parte a solicitação de cancelamento. Deverá ser um número inteiro entre 0 e 999.

destinatario
O *Destinatario* ou None.

class satcfe.entidades.**CFeVenda**(*emitente=None*, *destinatario=None*, *entrega=None*, *detalhamentos=None*, *descontos_acrescimos_subtotal=None*, *pagamentos=None*, *informacoes_adicionais=None*, ***kwargs*)
Representa um CF-e de venda.

Parâmetros

- **emitente** (Emitente) – Identificação do emitente do CF-e.
- **destinatario** (Destinatario) – Opcional. Identificação do destinatário.
- **entrega** (LocalEntrega) – Opcional. Informações do local de entrega.
- **detalhamentos** (*list*) – Uma lista de objetos *ProdutoServico* que representam os produtos/serviços participantes do CF-e de venda.
- **descontos_acrescimos_subtotal** (DescAcrEntr) – Opcional. Se informado, deverá ser um objeto *DescAcrEntr* que contenha o valor de desconto ou acréscimo sobre o subtotal.
- **pagamentos** (*list*) – Uma lista de objetos *MeioPagamento* que descrevem cada um dos meios de pagamentos usados no CF-e de venda.
- **informacoes_adicionais** (InformacoesAdicionais) – Opcional.

- **versaoDadosEnt** (*str*) – Opcional. String contendo a versão do layout do arquivo de dados do aplicativo comercial. Se não informado será utilizado o valor da constante VERSAO_LAYOUT_ARQUIVO_DADOS_AC do módulo constantes do *projeto ‘‘satcomum‘ <https://github.com/base4sistemas/satcomum/>‘_*
- **CNPJ** (*str*) – CNPJ da software house, desenvolvedora do aplicativo comercial, contendo apenas os dígitos do número e incluindo zeros não significativos, se for o caso (14 dígitos).
- **signAC** (*str*) – Assinatura do aplicativo comercial (344 dígitos).
- **numeroCaixa** (*int*) – Número do caixa ao qual o SAT está conectado. Normalmente este será o número do caixa de onde parte a solicitação de cancelamento. Deverá ser um número inteiro entre 0 e 999.
- **vCFeLei12741** (*Decimal*) – Opcional. Se informado deve representar a soma total dos valores aproximados dos tributos, em cumprimento à Lei nº 12.741/2012.

..note:

```
Não há uma classe específica para representar o elemento ``ide``
do grupo ``B01``, já que todos os seus atributos são esperados nesta
classe.
```

descontos_acrescimos_subtotal
Os descontos e acréscimos no subtotal do CF-e (*DescAcrEntr*) ou None.

destinatario
O *Destinatario* do CF-e ou None.

detalhamentos
Lista de objetos *Detalhamento*, descrevendo os produtos e serviços do CF-e.

emitente
O *Emitente* do CF-e.

entrega
O Local de entrega (*LocalEntrega*) ou None.

informacoes_adicionais
Informações adicionais do CF-e (*InformacoesAdicionais*) ou None.

pagamentos
Lista de objetos :class‘MeioPagamento‘, descrevendo os meios de pagamento empregados na quitação do CF-e.

class satcfe.entidades.**COFINSAliq**(***kwargs*)
Grupo de COFINS tributado pela alíquota, CST 01, 02 ou 05 (COFINSAliq, grupo S02).

Parâmetros

- **CST** (*str*) –
- **vBC** (*Decimal*) –
- **pCOFINS** (*Decimal*) –

class satcfe.entidades.**COFINSNT**(***kwargs*)
Grupo de COFINS não tributado, CST 04, 06, 07 08 ou 09 (COFINSNT, grupo S04).

Parâmetros **CST** (*str*) –

class satcfe.entidades.**COFINSOutr**(***kwargs*)
Grupo de COFINS para outras operações, CST 99 (COFINSOutr, grupo S06).

Parâmetros

- **CST** (*str*) –
- **vBC** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro pCOFINS.
- **pCOFINS** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro vBC.
- **qBCProd** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro vAliqProd.
- **vAliqProd** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro qBCProd.

Nota: Os parâmetros vBC e qBCProd são mutuamente exclusivos, e **um ou outro devem** ser informados.

---

class satcfe.entidades.**COFINSQtde**(***kwargs*)
Grupo de COFINS tributado por quantidade, CST 03 (COFINSQtde, grupo S03).

Parâmetros

- **CST** (*str*) –
- **qBCProd** (*Decimal*) –
- **vAliqProd** (*Decimal*) –

class satcfe.entidades.**COFINSSN**(***kwargs*)
Grupo de COFINS para contribuíntes do Simples Nacional, CST 49 (COFINSSN, grupo S05).

Parâmetros **CST** (*str*) –

class satcfe.entidades.**COFINSST**(***kwargs*)
Grupo de COFINS substituição tributária (COFINSST, grupo T01).

Parâmetros

- **vBC** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro pCOFINS.
- **pCOFINS** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro vBC.
- **qBCProd** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro vAliqProd.
- **vAliqProd** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro qBCProd.

Nota: Os parâmetros vBC e qBCProd são mutuamente exclusivos, e **um ou outro **devem** ser informados.

---

class satcfe.entidades.**DescAcrEntr**(***kwargs*)
Grupo de valores de entrada de desconto/acréscimo sobre subtotal (DescAcrEntr, grupo W19).

Parâmetros

- **vDescSubtot** (*Decimal*) – Valor de entrada de desconto sobre subtotal. Se este argumento for informado, então o argumento vAcresSubtot não deve ser informado.
- **vAcresSubtot** (*Decimal*) – Valor de entrada de acréscimo sobre subtotal. Se este argumento for informado, então o argumento vDescSubtot não deve ser informado.

class satcfe.entidades.**Destinatario**(***kwargs*)
Identificação do destinatário do CF-e (dest, grupo E01).

Parâmetros

- **CNPJ** (*str*) – Número do CNPJ do destinatário, contendo apenas os digitos e incluindo os zeros não significativos. **Não deve ser informado se o ‘‘CPF‘‘ for informado.**
- **CPF** (*str*) – Número do CPF do destinatário, contendo apenas os digitos e incluindo os zeros não significativos. **Não deve ser informado se o ‘‘CNPJ‘‘ for informado.**
- **xNome** (*str*) – Opcional. Nome ou razão social do destinatário. O nome do destinatário será ignorado no XML do CF-e de cancelamento.

class satcfe.entidades.**Detalhamento**(*produto=None*, *imposto=None*, ***kwargs*)
Detalhamento do produto ou serviço do CF-e (det, grupo H01).

Parâmetros

- **produto** (ProdutoServico) – Produto ou serviço, como uma instância de *ProdutoServico* ao qual o detalhamento se refere.
- **imposto** (Imposto) – O grupo de tributos incidentes no produto ou serviço ao qual o detalhamento se refere, como uma instância de *Imposto*.
- **infAdProd** (*str*) – Opcional. Informações adicionais do produto ou serviço (norma referenciada, informações complementares, etc).

Nota: O atributo XML nItem (H02) não é determinado aqui, mas atribuído automaticamente, conforme a sua posição na lista de *detalhamentos*.

---

class satcfe.entidades.**Emitente**(***kwargs*)
Identificação do emitente do CF-e (emit, grupo C01).

Parâmetros

- **CNPJ** (*str*) – Número do CNPJ do emitente do CF-e, contendo apenas os digitos e incluindo os zeros não significativos.
- **IE** (*str*) – Número de Inscrição Estadual do emitente do CF-e, contendo apenas digitos.
- **IM** (*str*) – Opcional. Deve ser informado o número da Inscrição Municipal quando o CF-e possuir itens com prestação de serviços sujeitos ao ISSQN, por exemplo.
- **cRegTribISSQN** (*str*) – Opcional. Indica o regime especial de tributação do ISSQN. Veja as constantes em C15_CREGTRIBISSQN_EMIT.
- **indRatISSQN** (*str*) – Opcional. Indicador de rateio do desconto sobre o subtotal entre itens sujeitos à tributação pelo ISSQN. Veja as constantes em C16_INDRATISSQN_EMIT.

class satcfe.entidades.**Entidade**(*schema={}*, *validator_class=None*, ***kwargs*)
Classe base para todas as classes que representem as entidades da implementação do SAT-CF-e. Aqui, chamaremos de “entidade” as classes que representem os grupos de dados que são usados para formar o XML do CF-e de venda ou de cancelamento.

Basicamente, as subclasses precisam sobrescrever a implementação do método _construir_elemento_xml, definir o atributo _schema e, quando necessário, implementar uma especialização do validador no atributo _validator_class.

documento(**args*, ***kwargs*)
Resulta no documento XML como string, que pode ou não incluir a declaração XML no início do documento.

class satcfe.entidades.**ExtendedValidator**(**args*, ***kwargs*)

Validator class. Normalizes and/or validates any mapping against a validation-schema which is provided as an argument at class instantiation or upon calling the `validate()`, `validated()` or `normalized()` method. An instance itself is callable and executes a validation.

All instantiation parameters are optional.

There are the introspective properties `types`, `validators`, `coercers`, `default_setters`, `rules`, `normalization_rules` and `validation_rules`.

The attributes reflecting the available rules are assembled considering constraints that are defined in the docstrings of rules' methods and is effectively used as validation schema for `schema`.

Parâmetros

- **schema** (any mapping) – See `schema`. Defaults to `None`.
- **ignore_none_values** (`bool`) – See `ignore_none_values`. Defaults to `False`.
- **allow_unknown** (`bool` or any mapping) – See `allow_unknown`. Defaults to `False`.
- **require_all** (`bool`) – See `require_all`. Defaults to `False`.
- **purge_unknown** (`bool`) – See `purge_unknown`. Defaults to to `False`.
- **purge_readonly** (`bool`) – Removes all fields that are defined as `readonly` in the normalization phase.
- **error_handler** (class or instance based on `BaseErrorHandler` or `tuple`) – The error handler that formats the result of `errors`. When given as two-value tuple with an error-handler class and a dictionary, the latter is passed to the initialization of the error handler. Default: `BasicErrorHandler`.

class satcfe.entidades.**ICMS00**(***kwargs*)

Grupo de tributação do ICMS 00, 20 e 90 (`ICMS00`, grupo `N02`).

Parâmetros

- **Orig** (*str*) –
- **CST** (*str*) –
- **pICMS** (*Decimal*) –

class satcfe.entidades.**ICMS40**(***kwargs*)

Grupo de tributação do ICMS 40, 41 e 60 (`ICMS40`, grupo `N03`).

Parâmetros

- **Orig** (*str*) –
- **CST** (*str*) –

class satcfe.entidades.**ICMSSN102**(***kwargs*)

Grupo de tributação do ICMS Simples Nacional, CSOSN 102, 300, 400 e 500 (`ICMSSN102`, grupo `N04`).

Parâmetros

- **Orig** (*str*) –
- **CSOSN** (*str*) –

class satcfe.entidades.**ICMSSN900**(***kwargs*)

Grupo de tributação do ICMS Simples Nacional, CSOSN 900 (`ICMSSN900`, grupo `N05`).

Parâmetros

- **Orig** (*str*) –
- **CSOSN** (*str*) –
- **pICMS** (*Decimal*) –

class satcfe.entidades.**ISSQN**(***kwargs*)

Grupo do ISSQN (ISSQN, grupo U01).

Parâmetros

- **vDeducISSQN** (*Decimal*) –
- **vAliq** (*Decimal*) –
- **cMunFG** (*str*) – Opcional.
- **cListServ** (*str*) – Opcional.
- **cServTribMun** (*str*) – Opcional.
- **cNatOp** (*str*) –
- **indIncFisc** (*str*) –

class satcfe.entidades.**Imposto**(*icms=None*, *pis=None*, *pisst=None*, *cofins=None*, *cofinsst=None*, *issqn=None*, ***kwargs*)

Grupo de tributos incidentes no produto ou serviço (imposto, grupo M01).

Parâmetros

- **icms** – Opcional. Deve ser uma instância de uma das classes dos grupos de ICMS (*ICMS00*, *ICMS40*, *ICMSSN102* ou *ICMSSN900*) se o item for um produto tributado pelo ICMS ou None em caso contrário.
- **pis** – Deve ser uma instância de uma das classes dos grupos de PIS (*PISAliq*, *PISQtde*, *PISNT*, *PISSN* ou *PISOutr*).
- **pisst** – Opcional. Instância de *PISST* ou None.
- **cofins** – Deve ser uma instância de uma das classes dos grupos de COFINS (*COFINSAliq*, *COFINSQtde*, *COFINSNT*, *COFINSSN* ou *COFINSOutr*).
- **cofinsst** – Opcional. Instância de *COFINSST* ou None.
- **issqn** – Opcional. Uma instância de *ISSQN* se o item for um serviço tributado pelo ISSQN ou None em caso contrário.
- **vItem12741** (*Decimal*) – Opcional. Valor aproximado dos tributos do produto ou serviço, conforme a Lei 12.741/12.

cofins

Um dos grupos de COFINS (*COFINSAliq*, *COFINSQtde*, *COFINSNT*, *COFINSSN* ou *COFINSOutr*).

cofinsst

O grupo do COFINS Substituição Tributária (*COFINSST*) se for o caso, ou None.

icms

Um dos grupos de ICMS (*ICMS00*, *ICMS40*, *ICMSSN102* ou *ICMSSN900*) se o item for um produto tributado pelo ICMS ou None em caso contrário.

issqn

O grupo de ISSQN (*ISSQN*) se o item for um serviço tributado pelo ISSQN ou None em caso contrário.

pis

Um dos grupos de PIS (*PISAliq*, *PISQtde*, *PISNT*, *PISSN* ou *PISOutr*).

pisst
O grupo do PIS Substituição Tributária (*PISST*) se for o caso, ou None.

class satcfe.entidades.**InformacoesAdicionais**(***kwargs*)
Grupo de informações adicionais (infAdic, grupo Z01).

Parâmetros **infCpl** (*str*) – Opcional.

class satcfe.entidades.**LocalEntrega**(***kwargs*)
Identificação do Local de Entrega (entrega, grupo G01).

Parâmetros

- **xLgr** (*str*) –
- **nro** (*str*) –
- **xCpl** (*str*) – Opcional.
- **xBairro** (*str*) –
- **xMun** (*str*) –
- **UF** (*str*) –

class satcfe.entidades.**MeioPagamento**(***kwargs*)
Meio de pagamento (MP, grupo WA02).

Parâmetros

- **cMP** (*str*) –
- **vMP** (*Decimal*) –
- **cAdmC** (*str*) – Opcional.

class satcfe.entidades.**ObsFiscoDet**(***kwargs*)
Grupo do campo de uso livre do Fisco (obsFiscoDet, grupo I17).

Parâmetros

- **xCampoDet** (*str*) –
- **xTextoDet** (*str*) –

class satcfe.entidades.**PISAliq**(***kwargs*)
Grupo de PIS tributado pela alíquota, CST 01, 02 ou 05 (PISAliq, grupo Q02).

Parâmetros

- **CST** (*str*) –
- **vBC** (*Decimal*) –
- **pPIS** (*Decimal*) –

class satcfe.entidades.**PISNT**(***kwargs*)
Grupo de PIS não tributado, CST 04, 06, 07 08 ou 09 (PISNT, grupo Q04).

Parâmetros **CST** (*str*) –

class satcfe.entidades.**PISOutr**(***kwargs*)
Grupo de PIS para outras operações, CST 99 (PISOutr, grupo Q06).

Parâmetros

- **CST** (*str*) –
- **vBC** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro pPIS.

- **pPIS** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro vBC.
- **qBCProd** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro vAliqProd.
- **vAliqProd** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro qBCProd.

Nota: Os parâmetros vBC e qBCProd são mutuamente exclusivos, e **um ou outro devem** ser informados.

class satcfe.entidades.**PISQtde**(***kwargs*)
Grupo de PIS tributado por quantidade, CST 03 (PISQtde, grupo Q03).

Parâmetros

- **CST** (*str*) –
- **qBCProd** (*Decimal*) –
- **vAliqProd** (*Decimal*) –

class satcfe.entidades.**PISSN**(***kwargs*)
Grupo de PIS para contribuíntes do Simples Nacional, CST 49 (PISSN, grupo Q05).

Parâmetros **CST** (*str*) –

class satcfe.entidades.**PISST**(***kwargs*)
Grupo de PIS substituição tributária (PISST, grupo R01).

Parâmetros

- **vBC** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro pPIS.
- **pPIS** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro vBC.
- **qBCProd** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro vAliqProd.
- **vAliqProd** (*str*) – Opcional. Se informado deverá ser também informado o parâmetro qBCProd.

Nota: Os parâmetros vBC e qBCProd são mutuamente exclusivos, e **um ou outro devem** ser informados.

class satcfe.entidades.**ProdutoServico**(*observacoes_fisco=None*, ***kwargs*)
Produto ou serviço do CF-e (prod, grupo I01).

Parâmetros

- **cProd** (*str*) –
- **cEAN** (*str*) – Opcional.
- **xProd** (*str*) –
- **NCM** (*str*) – Opcional.
- **CFOP** (*str*) –
- **uCom** (*str*) –
- **qCom** (*Decimal*) –
- **vUnCom** (*Decimal*) –

- **indRegra** (*str*) –
- **vDesc** (*Decimal*) – Opcional.
- **vOutro** (*Decimal*) – Opcional.
- **observacoes_fisco** (*list*) – Opcional. Lista de objetos *ObsFiscoDet* representando os campos de uso livre do fisco.

3.7.5 Módulo satcfe.excecoes

exception satcfe.excecoes.**ErroRespostaSATInvalida**
Lançada quando a resposta dada por uma função da DLL SAT não contém informação que faça sentido dentro do contexto. Este erro é diferente de uma *ExcecaoRespostaSAT* que é lançada quando a resposta faz sentido mas é interpretada como uma exceção a um comando que falhou.

exception satcfe.excecoes.**ExcecaoRespostaSAT**(*resposta*)
Lançada quando uma resposta à uma função da DLL SAT (comando SAT) é interpretada como tendo falhado. São casos em que a resposta é perfeitamente válida mas é interpretada como falha.

Por exemplo, quando a função `ConsultarSAT` é invocada e a resposta indica um código `EEEEE` diferente de `08000`, então uma exceção como esta será lançada.

3.7.6 Módulo satcfe.rede

class satcfe.rede.**ConfiguracaoRede**(***kwargs*)
Uma entidade que contém os parâmetros de configurações da interface de rede do equipamento SAT. Uma instância desta classe é usada como argumento para `configurar_interface_de_rede()`.

Parâmetros

- **tipoInter** (*str*) – Tipo de interface de rede que o equipamento SAT deverá utilizar. As opções de tipos de rede estão disponíveis na constante `REDE_TIPOINTER_OPCOES`.
- **SSID** (*str*) – Opcional. Nome da rede sem fio contendo até 32 caracteres.
- **seg** (*str*) – Opcional. Tipo de segurança da rede sem fio. As opções estão na constante `REDE_SEG_OPCOES`.
- **codigo** (*str*) – Opcional. Senha de acesso à rede sem fio, contendo até 64 caracteres.
- **tipoLan** (*str*) – Tipo da rede LAN. As opções estão disponíveis na constante `REDE_TIPOLAN_OPCOES`.
- **lanIP** (*str*) – Opcional. Endereço IP do equipamento SAT.
- **lanMask** (*str*) – Opcional. Máscara de sub-rede.
- **lanGW** (*str*) – Opcional. Endereço IP do gateway padrão.
- **lanDNS1** (*str*) – Opcional. Endereço IP do DNS primário.
- **lanDNS2** (*str*) – Opcional. Endereço IP do DNS secundário.
- **usuario** (*str*) – Opcional. Nome do usuário para obtenção do endereço IP, se necessário, contendo até 64 caracteres.
- **senha** (*str*) – Opcional. Senha do usuário para obtenção do endereço IP, relacionado ao parâmetro `usuario`, se necessário, contendo até 32 caracteres.
- **proxy** (*str*) – Opcional. Indica a configuração de proxy da rede. As opções estão disponíveis na constante `REDE_PROXY_OPCOES`.

- **proxy_ip** (*str*) – Opcional. Endereço IP do servidor proxy.
- **proxy_porta** (*int*) – Opcional. Número da porta por onde o servidor de proxy responde.
- **proxy_user** (*str*) – Opcional. Nome do usuário para acesso ao proxy, se necessário, contendo até 64 caracteres.
- **proxy_senha** (*str*) – Opcional. Senha do usuário para acesso ao proxy, relacionado ao parâmetro `proxy_user`, se necessário, contendo até 64 caracteres.

3.7.7 Módulo `satcfe.util`

satcfe.util.**as_date**(*value*)
Converte uma sequência string para um objeto `datetime.date`. Os espaços em branco das bordas da sequência serão removidos antes da conversão.

Parâmetros **value** (*str*) – String contendo uma data ANSI (`yyyymmdd`)

Tipo de retorno datetime.date

satcfe.util.**as_date_or_none**(*value*)
Converte uma sequência string para um objeto `datetime.date` ou resulta em `None` se a sequência não for uma data válida. Os espaços em branco das bordas da sequência serão removidos antes da conversão.

Parâmetros **value** (*str*) – String contendo uma data ANSI (`yyyymmdd`)

Tipo de retorno datetime.date or None

satcfe.util.**as_datetime**(*value*)
Converte uma sequência string para um objeto `datetime.datetime`. Os espaços em branco das bordas da sequência serão removidos antes da conversão.

Parâmetros **value** (*str*) – String contendo uma data/hora ANSI (`yyyymmddHHMMSS`)

Tipo de retorno datetime.datetime

satcfe.util.**as_datetime_or_none**(*value*)
Converte uma sequência string para um objeto `datetime.datetime` ou resulta em `None` se a sequência não for uma data/hora válidas. Os espaços em branco das bordas da sequência serão removidos antes da conversão.

Parâmetros **value** (*str*) – String contendo uma data/hora ANSI (`yyyymmddHHMMSS`)

Tipo de retorno datetime.datetime or None

satcfe.util.**base64_to_str**(*data*)
Decodifica uma massa de dados codificada em Base64.

Parâmetros **data** (*str*) – String contendo a massa de dados codificada em Base64.

Tipo de retorno str

satcfe.util.**hms**(*segundos*)
Retorna o número de horas, minutos e segundos a partir do total de segundos informado.

Parâmetros **segundos** (*int*) – O número total de segundos.

Retorna Uma tupla contendo trs elementos representando, respectivamente, o número de horas, minutos e segundos calculados a partir do total de segundos.

Tipo de retorno tuple

satcfe.util.**hms_humanizado**(*segundos*)
Retorna um texto legível, amigável, que descreve o total de horas, minutos e segundos calculados a partir do total de segundos informados.

Parâmetros **segundos** (*int*) – O número total de segundos.

Tipo de retorno str

satcfe.util.**normalizar_ip**(*ip*)
Normaliza uma sequência string que contenha um endereço IPv4.

Normalmente os equipamentos SAT, seguindo a ER SAT, resultam endereços IP com um aspecto similar a `010.000.000.001`, visualmente desagradável e difícil de ler. Esta função normaliza o endereço acima como `10.0.0.1`.

Parâmetros **ip** (*str*) – String contendo um endereço IPv4.

Tipo de retorno str

satcfe.util.**str_to_base64**(*data*, *encoding='utf-8'*)
Codifica uma string (por padrão, UTF-8) em Base64.

Parâmetros

- **data** (*str*) – String a ser codificada em Base64.
- **encoding** (*str*) – Opcional. O *encoding* da string *data*. Se não for especificado, o padrão é *'utf-8'*.

Retorna Uma string UTF-8 contendo a massa de dados em Base64.

Tipo de retorno str

3.7.8 Respostas das Funções SAT

As funções da biblioteca SAT retornam sequências de texto que contém os atributos da resposta. Os atributos estão separados entre si por um caracter de *linha vertical*, ou pipe.

```
567102|09000|Emitido com sucesso||
```

As classes *ClienteSATLocal* e *ClienteSATHub* resultam em respostas que são objetos Python que facilitam o acesso à esses atributos, mesmo quando a comunicação com o equipamento foi bem sucedida mas a resposta indica um erro. Veja como lidar com algumas das respostas mais básicas em *Funções Básicas e de Consulta* e *Lidando com Exceções*.

Módulo satcfe.resposta.padrao

class satcfe.resposta.padrao.**RespostaSAT**(***kwargs*)
Base para representação de respostas das funções da biblioteca SAT. A maior parte das funções SAT resultam em respostas que contém um conjunto padrão de atributos (veja o atributo *CAMPOS*), descritos na ER SAT:

```
numeroSessao (int)
EEEEE (text)
mensagem (text)
cod (text)
mensagemSEFAZ (text)
```

Além dos atributos padrão, a resposta deverá conter uma referência para o nome da função SAT a que a resposta se refere e ao conteúdo original da resposta, através dos atributos:

```
resposta.atributos.funcao
resposta.atributos.verbatim
```

Esta classe fornece uma série de métodos construtores (*factory methods*) para respostas que são comuns. Para as respostas que não são comuns, existem especializações desta classe.

Nota: Espera-se que a resposta original, devolvida pela biblioteca do fabricante do equipamento SAT, será sempre um dado Unicode.

Nota: Aqui, `text` diz respeito à um objeto `unicode` (Python 2) ou `str` (Python 3). Veja `builtins.str` da biblioteca `future`.

class Atributos(*funcao=None*, *verbatim=None*)

funcao

verbatim

CAMPOS = (('numeroSessao', <class 'int'>), ('EEEEE', <class 'str'>), ('mensagem', <cla
Campos padrão esperados em uma resposta e a sua função de conversão para o tipo Python, a partir da resposta original.

static atualizar_software_sat(*retorno*)
Constrói uma *RespostaSAT* para o retorno da função *atualizar_software_sat()*.

static bloquear_sat(*retorno*)
Constrói uma *RespostaSAT* para o retorno da função *bloquear_sat()*.

static comunicar_certificado_icpbrasil(*retorno*)
Constrói uma *RespostaSAT* para o retorno da função *comunicar_certificado_icpbrasil()*.

static configurar_interface_de_rede(*retorno*)
Constrói uma *RespostaSAT* para o retorno da função *configurar_interface_de_rede()*.

static consultar_sat(*retorno*)
Constrói uma *RespostaSAT* para o retorno da função *consultar_sat()*.

static desbloquear_sat(*retorno*)
Constrói uma *RespostaSAT* para o retorno da função *desbloquear_sat()*.

static trocar_codigo_de_ativacao(*retorno*)
Constrói uma *RespostaSAT* para o retorno da função *trocar_codigo_de_ativacao()*.

satcfe.resposta.padrao.**analisar_retorno**(*retorno*, *classe_resposta=<class 'satcfe.resposta.padrao.RespostaSAT'>*, *campos=(('numeroSessao', <class 'int'>), ('EEEEE', <class 'str'>), ('mensagem', <class 'str'>), ('cod', <class 'str'>), ('mensagemSEFAZ', <class 'str'>))*, *campos_alternativos=[]*, *funcao=None*, *manter_verbatim=True*)
Analisa o retorno (supostamente um retorno de uma função do SAT) conforme o padrão e campos esperados. O retorno deverá possuir dados separados entre si através de pipes e o número de campos deverá coincidir com os campos especificados.

Parâmetros

- **retorno** (*str*) – O conteúdo da resposta retornada pela função da biblioteca do fabricante do equipamento SAT, que espera-se que seja um dado Unicode.
- **classe_resposta** (*type*) – O tipo *RespostaSAT* ou especialização que irá representar o retorno, após sua decomposição em campos.
- **campos** (*tuple*) – Especificação dos campos e seus conversores. Os campos devem ser especificados como uma tupla onde cada elemento deverá ser uma tupla contendo dois elementos: o nome do campo e uma função de conversão a partir de uma string.
- **campos_alternativos** (*list*) – Uma lista de campos alternativos que serão considerados caso o número de campos encontrados na resposta não coincida com o número de campos do argumento `campos`. Para que a relação alternativa de campos funcione, é importante que cada relação de campos alternativos tenha um número diferente de campos.
- **funcao** (*str*) – Nome da função da DLL SAT que gerou o retorno, que estará disponível nos atributos adicionais à resposta.
- **manter_verbatim** (*bool*) – Se uma cópia verbatim da resposta deverá ser mantida nos atributos adicionais à resposta.

Levanta ***ErroRespostaSATInvalida*** – Se o retorno não estiver em conformidade com o padrão esperado ou se não possuir os campos especificados.

Retorna Uma instância de *RespostaSAT* ou especialização.

Tipo de retorno *satcfe.resposta.padrao.RespostaSAT*

Módulo `satcfe.resposta.ativarsat`

class satcfe.resposta.ativarsat.**RespostaAtivarSAT**(***kwargs*)

Lida com as respostas da função `AtivarSAT` (veja o método *ativar_sat()*). Os atributos esperados em caso de sucesso, são:

```
numeroSessao (int)
EEEEE (text)
mensagem (text)
cod (text)
mensagemSEFAZ (text)
CSR (text)
```

Em caso de falha, são esperados apenas os atributos padrão, conforme descrito na constante *CAMPOS*.

Nota: Aqui, `text` diz respeito à um objeto `unicode` (Python 2) ou `str` (Python 3). Veja `builtins.str` da biblioteca `future`.

---

static analisar(*retorno*)

Constrói uma *RespostaAtivarSAT* a partir do retorno informado.

Parâmetros **retorno** (*str*) – Retorno da função `AtivarSAT`.

csr()

Retorna o CSR (**Certificate Signing Request**) decodificado.

Módulo satcfe.resposta.cancelarultimavenda

class satcfe.resposta.cancelarultimavenda.**RespostaCancelarUltimaVenda**(***kwargs*)

Lida com as respostas da função CancelarUltimaVenda (veja o método *cancelar_ultima_venda()*). Os atributos esperados em caso de sucesso, são:

```
numeroSessao (int)
EEEEE (text)
CCCC (text)
mensagem (text)
cod (text)
mensagemSEFAZ (text)
arquivoCFeBase64 (text)
timeStamp (datetime.datetime)
chaveConsulta (text)
valorTotalCFe (decimal.Decimal)
CPFCNPJValue (text)
assinaturaQRCODE (text)
```

Em caso de falha, são esperados apenas os atributos:

```
numeroSessao (int)
EEEEE (text)
CCCC (text)
mensagem (text)
cod (text)
mensagemSEFAZ (text)
```

Finalmente, como último recurso, a resposta poderá incluir apenas os atributos padrão, conforme descrito na constante *CAMPOS*.

Nota: Aqui, text diz respeito à um objeto unicode (Python 2) ou str (Python 3). Veja builtins.str da biblioteca future.

---

static analisar(*retorno*)

Constrói uma *RespostaCancelarUltimaVenda* a partir do retorno informado.

Parâmetros **retorno** (*str*) – Retorno da função CancelarUltimaVenda.

qrcode()

Resulta nos dados que compõem o QRCode.

Tipo de retorno str

xml()

Retorna o XML do CF-e-SAT de cancelamento decodificado.

Tipo de retorno str

Módulo satcfe.resposta.consultarnumerosessao

class satcfe.resposta.consultarnumerosessao.**RespostaConsultarNumeroSessao**(***kwargs*)

Lida com as respostas da função ConsultarNumeroSessao (veja o método *consultar_numero_sessao()*). Como as respostas dependem do número da sessão consultado, o método de construção *analisar()* deverá resultar na resposta apropriada para cada retorno.

static analisar(*retorno*)
Constrói uma `RespostaSAT` ou especialização dependendo da função SAT encontrada na sessão consultada.

Parâmetros **retorno** (*str*) – Retorno da função `ConsultarNumeroSessao`.

Módulo satcfe.resposta.consultarstatusoperacional

satcfe.resposta.consultarstatusoperacional.**ESTADOS_OPERACAO = ((0, 'Desbloqueado'), (1, 'B**
Códigos do estados de operação e suas descrições amigáveis.

class satcfe.resposta.consultarstatusoperacional.**RespostaConsultarStatusOperacional**(***kwargs*
Lida com as respostas da função `ConsultarStatusOperacional` (veja o método *consultar_status_operacional()*). Os atributos esperados em caso de sucesso, são:

Atributo	Tipo Python
numeroSessao	int
EEEEE	text
mensagem	text
cod	text
mensagemSEFAZ	text
NSERIE	text
TIPO_LAN	text
LAN_IP	text
LAN_MAC	text
LAN_MASK	text
LAN_GW	text
LAN_DNS_1	text
LAN_DNS_2	text
STATUS_LAN	text
NIVEL_BATERIA	text
MT_TOTAL	text
MT_USADA	text
DH_ATUAL	datetime.datetime
VER_SB	text
VER_LAYOUT	text
ULTIMO_CF_E_SAT	text
LISTA_INICIAL	text
LISTA_FINAL	text
DH_CFE	datetime.datetime``|``None
DH_ULTIMA	datetime.datetime
CERT_EMISSAO	datetime.date
CERT_VENCIMENTO	datetime.date
ESTADO_OPERACAO	int

Em caso de falha, são esperados apenas os atributos padrão, conforme descrito na constante *CAMPOS*.

Nota: Aqui, `text` diz respeito à um objeto `unicode` (Python 2) ou `str` (Python 3). Veja `builtins.str` da biblioteca `future`.

---

static analisar(*retorno*)
Constrói uma *RespostaConsultarStatusOperacional* a partir do retorno informado.

Parâmetros **retorno** (*str*) – Retorno da função `ConsultarStatusOperacional`.

status
Nome amigável do campo `ESTADO_OPERACAO`, conforme a "Tabela de Informações do Status do SAT".

Módulo satcfe.resposta.enviardadosvenda

class satcfe.resposta.enviardadosvenda.**RespostaEnviarDadosVenda**(***kwargs*)
Lida com as respostas da função `EnviarDadosVenda` (veja o método *enviar_dados_venda()*). Os atributos esperados em caso de sucesso, são:

```
numeroSessao (int)
EEEEE (text)
CCCC (text)
mensagem (text)
cod (text)
mensagemSEFAZ (text)
arquivoCFeSAT (text)
timeStamp (datetime.datetime)
chaveConsulta (text)
valorTotalCFe (decimal.Decimal)
CPFCNPJValue (text)
assinaturaQRCODE (text)
```

Em caso de falha, são esperados apenas os atributos:

```
numeroSessao (int)
EEEEE (text)
CCCC (text)
mensagem (text)
cod (text)
mensagemSEFAZ (text)
```

Finalmente, como último recurso, a resposta poderá incluir apenas os atributos padrão, conforme descrito na constante *CAMPOS*.

Nota: Aqui, `text` diz respeito à um objeto `unicode` (Python 2) ou `str` (Python 3). Veja `builtins.str` da biblioteca `future`.

---

static analisar(*retorno*)
Constrói uma *RespostaEnviarDadosVenda* a partir do retorno informado.

Parâmetros **retorno** (*str*) – Retorno da função `EnviarDadosVenda`.

qrcode()
Resulta nos dados que compõem o QRCode.

Tipo de retorno str

xml()
Retorna o XML do CF-e-SAT decodificado de Base64.

Tipo de retorno str

Módulo satcfe.resposta.extrairlogs

class satcfe.resposta.extrairlogs.**RespostaExtrairLogs**(***kwargs*)

Lida com as respostas da função ExtrairLogs (veja o método *extrair_logs()*). Os atributos esperados em caso de sucesso, são:

```
numeroSessao (int)
EEEEE (text)
mensagem (text)
cod (text)
mensagemSEFAZ (text)
arquivoLog (text)
```

Em caso de falha, são esperados apenas os atributos padrão, conforme descrito na constante *CAMPOS*.

Nota: Aqui, text diz respeito à um objeto unicode (Python 2) ou str (Python 3). Veja builtins.str da biblioteca future.

---

static analisar(*retorno*)

Constrói uma *RespostaExtrairLogs* a partir do retorno informado.

Parâmetros **retorno** (*str*) – Retorno da função ExtrairLogs.

conteudo()

Retorna o conteúdo do log decodificado.

salvar(*destino=None*, *prefix='tmp'*, *suffix='-sat.log'*, *dir=None*, *encoding='utf-8'*, *encoding_errors='strict'*)

Salva o arquivo de log decodificado.

Parâmetros

- **destino** (*str*) – Opcional. Caminho completo para o arquivo onde os dados dos logs deverão ser salvos. Se não informado, será criado um arquivo temporário via tempfile.mkstemp().
- **prefix** (*str*) – Opcional. Prefixo para o nome do arquivo. Se não informado será usado "tmp".
- **suffix** (*str*) – Opcional. Sufixo para o nome do arquivo. Se não informado será usado "-sat.log".
- **dir** – Opcional. Contém o caminho completo onde o arquivo temporário deverá ser criado. Este argumento terá efeito apenas quando o argumento destino não for informado.
- **encoding** (*str*) – Opcional. Codificação de caracteres a ser usada para codificar o conteúdo do log em bytes que serão efetivamente escritos no arquivo de destino. Padrão é "utf-8". Veja o método str.encode() para detalhes.
- **encoding_errors** (*str*) – Opcional. Como lidar com os erros de codificação de caracteres. Padrão é "strict". Veja o método str.encode() para detalhes.

Retorna Retorna o caminho completo para o arquivo salvo.

Tipo de retorno str

Levanta **FileExistsError** – Se o destino informado já existir.

Módulo satcfe.resposta.testefimafim

class satcfe.resposta.testefimafim.**RespostaTesteFimAFim**(***kwargs*)

Lida com as respostas da função TesteFimAFim (veja o método *teste_fim_a_fim()*). Os atributos esperados em caso de sucesso, são:

```
numeroSessao (int)
EEEEE (text)
mensagem (text)
cod (text)
mensagemSEFAZ (text)
arquivoCFeBase64 (text)
timeStamp (datetime.datetime)
numDocFiscal (int)
chaveConsulta (text)
```

Em caso de falha, são esperados apenas os atributos padrão, conforme descrito na constante *CAMPOS*.

Nota: Aqui, text diz respeito à um objeto unicode (Python 2) ou str (Python 3). Veja builtins.str da biblioteca future.

---

static analisar(*retorno*)

Constrói uma *RespostaTesteFimAFim* a partir do retorno informado.

Parâmetros **retorno** (*str*) – Retorno da função TesteFimAFim.

Levanta ***ExcecaoRespostaSAT*** – Se o atributo EEEEE não indicar o código de sucesso 09000 para TesteFimAFim.

qrcode()

Resulta nos dados que compõem o QRCode.

Tipo de retorno str

xml()

Retorna o XML do CF-e-SAT decodificado.

Tipo de retorno str

3.7.9 API da Infraestrutura de Alertas

Fornece um mecanismo simples e robusto para checagem de potenciais problemas de operação com base no status operacional do equipamento SAT. Veja também a documentação de introdução da *Infraestrutura de Alertas*.

Módulo satcfe.alertas

class satcfe.alertas.**AlertaCFePendentes**(*resposta*)

Checa a existência de documentos CF-e-SAT pendentes no equipamento SAT, aguardando serem enviados à SEFAZ. Este alerta estará ativo se houver ao menos um documento CF-e-SAT pendente de transmissão no equipamento SAT.

checar()

Efetivamente checa se o alerta deve ou não ser ativado em função dos dados da resposta e de outras condições. As classes de alertas devem sobrescrever este método.

Retorna Retorna `True` se o resultado da checagem indicar que este alerta está ativo (o mesmo que `ativo`).

Tipo de retorno bool

mensagem()
Retorna uma mensagem amigável ao usuário, descrevendo da melhor forma possível a condição do alerta. As classes de alertas devem sobrescrever este método.

Tipo de retorno unicode

pendentes
Retorna o número de cupons pendentes de transmissão para a SEFAZ.

Tipo de retorno int

class satcfe.alertas.**AlertaDivergenciaHorarios**(*resposta*)
Checa o horário do equipamento SAT em relação ao horário atual, emitindo um alerta caso exista uma divergência entre os horáros superior a 3600 segundos (1 hora). Para alterar o limite de tolerância que ativará este alerta, modifique o atributo *tolerancia_em_segundos*.

Nota: O limite de tolerância para este alerta, de uma hora, é uma herança do **Requisito XVII** do PAF-ECF, *Sincronismo entre data e hora do registro com data e hora do Cupom Fiscal*, embora SAT-CF-e não tenha qualquer relação com o PAF-ECF.

---

checar()
Efetivamente checa se o alerta deve ou não ser ativado em função dos dados da resposta e de outras condições. As classes de alertas devem sobrescrever este método.

Retorna Retorna `True` se o resultado da checagem indicar que este alerta está ativo (o mesmo que `ativo`).

Tipo de retorno bool

divergencia
Divergência em segundos entre o horário local (do computador) e o horário do equipamento SAT, segundo a resposta de consulta ao status operacional. Precisão de microsegundos é desprezada.

Uma **divergência negativa** indica que o horário local (do computador) está atrasado em relação ao relógio do equipamento SAT. Para saber se a divergência de horarários ultrapassou o limite de tolerância, consulte o atributo *ativo*.

Tipo de retorno int

mensagem()
Retorna uma mensagem amigável ao usuário, descrevendo da melhor forma possível a condição do alerta. As classes de alertas devem sobrescrever este método.

Tipo de retorno unicode

tolerancia_em_segundos = 3600
Limite de tolerância, em segundos, para ativar o alerta.

class satcfe.alertas.**AlertaOperacao**(*resposta*)
Classe base para os alertas de operação.

ativo
Indica se o alerta está ou não ativo.

checar()
Efetivamente checa se o alerta deve ou não ser ativado em função dos dados da resposta e de outras condições. As classes de alertas devem sobrescrever este método.

Retorna Retorna `True` se o resultado da checagem indicar que este alerta está ativo (o mesmo que *ativo*).

Tipo de retorno bool

mensagem()
Retorna uma mensagem amigável ao usuário, descrevendo da melhor forma possível a condição do alerta. As classes de alertas devem sobrescrever este método.

Tipo de retorno unicode

class satcfe.alertas.**AlertaVencimentoCertificado**(*resposta*)
Checa a data de vencimento do certificado instalado, ativando o alerta caso o vencimento esteja próximo. Para alterar o limite de proximidade do vencimento que ativa este alerta, modifique o atributo *vencimento_em_dias*, cujo padrão é de 60 dias.

checar()
Efetivamente checa se o alerta deve ou não ser ativado em função dos dados da resposta e de outras condições. As classes de alertas devem sobrescrever este método.

Retorna Retorna `True` se o resultado da checagem indicar que este alerta está ativo (o mesmo que `ativo`).

Tipo de retorno bool

dias_para_vencimento
O número de dias que restam até o vencimento do certificado instalado. Se o certificado já estiver vencido, retornará zero.

Tipo de retorno int

mensagem()
Retorna uma mensagem amigável ao usuário, descrevendo da melhor forma possível a condição do alerta. As classes de alertas devem sobrescrever este método.

Tipo de retorno unicode

vencido
Indica se o certificado instalado no equipamento está vencido.

Tipo de retorno bool

vencimento_em_dias = 60
Determina o número de dias até o vencimento do certificado que irá ativar o alarte.

satcfe.alertas.**checar**(*cliente_sat*)
Checa em sequência os alertas registrados (veja *registrar()*) contra os dados da consulta ao status operacional do equipamento SAT. Este método irá então resultar em uma lista dos alertas ativos.

Parâmetros **cliente_sat** – Uma instância de *satcfe.clientelocal.ClienteSATLocal* ou *satcfe.clientesathub.ClienteSATHub* onde será invocado o método para consulta ao status operacional do equipamento SAT.

Tipo de retorno list

satcfe.alertas.**registrar**(*classe_alerta*)
Registra uma classe de alerta (subclasse de *AlertaOperacao*). Para mais detalhes, veja *checar()*.

3.8 Executando Testes

É possível executar os testes contra qualquer equipamento SAT, em qualquer plataforma ou arquitetura, desde que você possua um kit de desenvolvimento, contendo o equipamento SAT e as bibliotecas do fabricante.

Para executar os testes em um ambiente Linux é preciso definir algumas variáveis de ambiente para configurar o acesso à biblioteca SAT fornecida pelo fabricante do equipamento, o código de ativação e o Estado do domicílio fiscal em que o equipamento SAT está registrado.

```
$ export SATCFE_TEST_LIB=/opt/fabricante/libsat.so
$ export SATCFE_TEST_LIB_CONVENCAO=1
$ export SATCFE_TEST_CODIGO_ATIVACAO=12345678
$ export SATCFE_TEST_UF=SP
```

Antes de executar os testes propriamente, é conveniente revisar a parametrização no script `runtests.sh` que, dependendo do seu equipamento SAT, os valores para configuração dos dados do emitente e outros dados podem variar. Isto irá executar os testes invocando as funções `ConsultarSAT` e `ConsultarStatusOperacional` (revise o script para adicionar ou remover funções a serem invocadas):

```
$ ./runtests.sh tanca
```

3.8.1 Parametrização

As opções de parametrização dos testes são:

`--codigo-ativacao`

Código de ativação configurado no equipamento SAT.

`--numero-caixa`

Número do caixa de origem.

`--assinatura-ac`

Conteúdo da assinatura da AC.

`--cnpj-ac`

CNPJ da empresa desenvolvedora da AC (apenas dígitos).

`--emitente-cnpj`

CNPJ do estabelecimento emitente (apenas dígitos).

`--emitente-ie`

Inscrição estadual do emitente (apenas dígitos).

`--emitente-im`

Inscrição municipal do emitente (apenas dígitos).

`--emitente-uf`

Sigla da unidade federativa do estabelecimento emitente.

`--emitente-issqn-regime`

Regime especial de tributação do ISSQN do emitente, em casos de testes de emissão de venda e/ou cancelamento.

--emitente-issqn-rateio

Indicador de rateio do desconto sobre o subtotal para produtos tributados no ISSQN do emitente, em casos de testes de emissão de venda e/ou cancelamento.

--lib-caminho

Caminho para a biblioteca SAT.

--lib-convencao

Convenção de chamada para a biblioteca SAT.

--acessa-sat

Permite que sejam executados os testes que acessem a biblioteca SAT, eventualmente acessando o equipamento SAT real

--invoca-[funcao]

Permite que sejam executados os testes que acessem a biblioteca SAT, eventualmente acessando o equipamento SAT real, para acesso à função especificada (`funcao`). Por exemplo, `--invoca-consultarsat`.

3.8.2 Executando Testes Manualmente

Você poderá executar os testes unitários contra uma biblioteca de simulação que acompanha o projeto, chamada **mockuplib**. Isso facilita a execução dos testes unitários sem correr riscos de executar comandos sensíveis (como troca de código de ativação) em um equipamento real, mesmo que seja um equipamento específico para desenvolvimento.

Você irá precisar das ferramentas GNU Make e GNU GCC para compilar a biblioteca de simulação e então usar Pipenv e tox para executar os testes unitários:

```
$ make mockuplib
$ pipenv install --dev --clear
$ pipenv run tox
```

Dê uma olhada no arquivo `tox.ini` e procure pela propriedade `envlist`, que relaciona as versões de Python que serão usadas nos testes. Se você quiser executar os testes contra uma versão específica de Python, utilize a opção `-e`. Por exemplo, para executar os testes com a versão 3.6 de Python:

```
$ pipenv run tox -e py36
```

3.8.3 Testando Funções Específicas

Se não quiser (ou não puder) usar o script `runtests.sh` por alguma razão, você poderá comandar a execução dos testes unitários e dos testes que acessam a biblioteca SAT e invocam funções específicas (você terá que especificar uma por uma). Por exemplo, para executar o teste da função `ConsultarSAT` faça:

```
$ pipenv run python setup.py test -a "--acessa-sat --invoca-consultarsat"
```

3.8.4 Executando Testes usando GNU Make

Também é possível executar os testes (e outras tarefas) usando o `Makefile` que acompanha o projeto. Por exemplo para executar os testes que **não** acessam as funções da biblioteca SAT, faça:

```
$ pipenv shell
$ make test
```

Para executar todos os testes, **inclusive os testes contra a biblioteca SAT**, use o alvo `testall`. Esse alvo irá também compilar a biblioteca SAT *mockup* que acompanha o projeto justamente para execução completa dos testes, sem o risco de acessar um equipamento SAT. De qualquer maneira, mesmo utilizando a biblioteca *mockup* ou qualquer outra biblioteca SAT, é preciso definir a variável de ambiente `SATCFE_TEST_LIB` que deve apontar para a biblioteca SAT que será utilizada nos testes, por exemplo:

```
$ export SATCFE_TEST_LIB=satcfe/tests/mockup/libmockupsat.so
$ make testall
```

3.8.5 Variáveis de Ambiente para os Testes

Estas são todas as variáveis de ambiente utilizadas no script `runtests.sh` e usadas como valor padrão quando os testes são invocados manualmente (e seus valores padrão):

Variável	Valor Padrão
SATCFE_TEST_LIB	libsat.so
SATCFE_TEST_LIB_CONVENCAO	1[1]
SATCFE_TEST_CODIGO_ATIVACAO	12345678[2]
SATCFE_TEST_EMITENTE_UF	SP[2]
SATCFE_TEST_CNPJ_AC	16716114000172[2]
SATCFE_TEST_EMITENTE_CNPJ	08723218000186[2]
SATCFE_TEST_EMITENTE_IE	149626224113[2]
SATCFE_TEST_EMITENTE_IM	123123[2]
SATCFE_TEST_EMITENTE_ISSQN_REGIME	3[3]
SATCFE_TEST_EMITENTE_ISSQN_RATEIO	N[4]

[1] Veja constante `CONVENCOES_CHAMADA` no projeto SATComum para conhecer os valores possíveis.

[2] Os valores padrão são para equipamentos SAT de desenvolvimento fabricados pela Tanca. Se o seu equipamento for de um fabricante diferente substitua pelos valores indicados no manual. O script `runtests.sh` tem os valores padrão para alguns outros fabricantes, mas observe que esses valores podem mudar entre os modelos de um mesmo fabricante.

[3] Veja constante `C15_CREGTRIBISSQN_EMIT` no projeto SATComum para conhecer os valores possíveis.

[4] Veja constante `C16_INDRATISSQN_EMIT` no projeto SATComum para conhecer os valores possíveis.

CAPÍTULO 4

Tabelas e Índices

- genindex
- modindex
- search

CAPÍTULO 5

Glossário

AC

PDV

Ponto-de-Venda

Frente-de-Caixa Software capaz de realizar vendas e cancelamentos, gerando os detalhes da venda ou do cancelamento e cuidando de vários outros aspectos como pagamentos, por exemplo, além de toda a lógica de negócios, conforme os ramo de atividade do estabelecimento usuário. Este é o aplicativo cliente típico deste projeto.

AC-SAT Refere-se à **Autoridade Certificadora** que gerencia (emite e revoga) certificados digitais, que contém a chave criptográfica necessária para assinar digitalmente os documentos XML tornando-os documentos fiscais juridicamente válidos.

CF-e

CF-e de Venda

CF-e de Cancelamento Cupom Fiscal eletrônico, um documento em formato XML que descreve uma transação de venda ao consumidor ou o cancelamento de uma venda anterior. O **CF-e de Venda**, como o nome sugere, descreve uma venda completa, com seus produtos e quantidades, valores, impostos, meios de pagamento e observações. O **CF-e de Cancelamento** é um documento eletrônico muito parecido em sua estrutura com o CF-e de Venda, mas que documenta um cancelamento de uma venda feita anteriormente.

CF-e-SAT Refere-se ao CF-e (de venda ou de cancelamento) que transitou através do SAT-CF-e, ou seja, **é um documento fiscal com validade jurídica** e, o que o torna válido juridicamente é a assinatura digital que ele contém, e que o torna um documento único. Trata-se de um documento fiscal eletrônico autorizado pela SEFAZ.

ECF-IF Emissor de Cupons Fiscais - Impressora Fiscal. Basicamente uma mini-impressora acoplada à uma placa fiscal que emite cupons fiscais com validade jurídica e outros documentos não fiscais. No Estado de São Paulo, a tecnologia dos ECF-IF foi substituída, a partir de 2015, pelo *SAT-CF-e*.

Equipamento SAT Hardware responsável por receber, validar, assinar e transmitir os documentos XML que representam vendas ou cancelamentos. O equipamento também é responsável pelo modelo de contingência de operação, quando não é possível que seja estabelecida comunicação com a SEFAZ por qualquer razão, entre outras funções importantes.

ER SAT Especificação de Requisitos do SAT. É o documento oficial, escrito e mantido pela SEFAZ, que detalha a tecnologia SAT-CF-e do ponto de vista dos fabricantes dos equipamentos SAT e das empresas de software que desenvolvem os aplicativos comerciais. Note que a ER SAT **não é a legislação** que introduz o SAT-CF-e. A legislação é a CAT 147 de 05 de novembro de 2012.

SAT-CF-e Diz respeito à tecnologia SAT-Fiscal e toda a infraestrutura, física e lógica, usada na transmissão de documentos fiscais (CF-e) de venda e/ou cancelamento. Visite a página da Secretaria da Fazenda de São Paulo para outras informações.

CAPÍTULO 6

Créditos

Imagens criadas com Inkscape e editadas usando GIMP. Foram usadas as famílias de fontes de tipos Ubuntu Font Family e Font Awesome.

Índice de Módulos Python

S

Índice

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

V

X