SATCFe Documentation

Versão 0.0.7

Base4 Sistemas Ltda

02/10/2015

Sumário

This project is about **SAT-CF-e** which is a system for autorization and transmission of fiscal documents, developed by Finance Secretary of state of São Paulo, Brazil. This entire project, variables, methods and class names, as well as documentation, are written in brazilian portuguese.

Refer to the oficial web site for more information (in brazilian portuguese only).

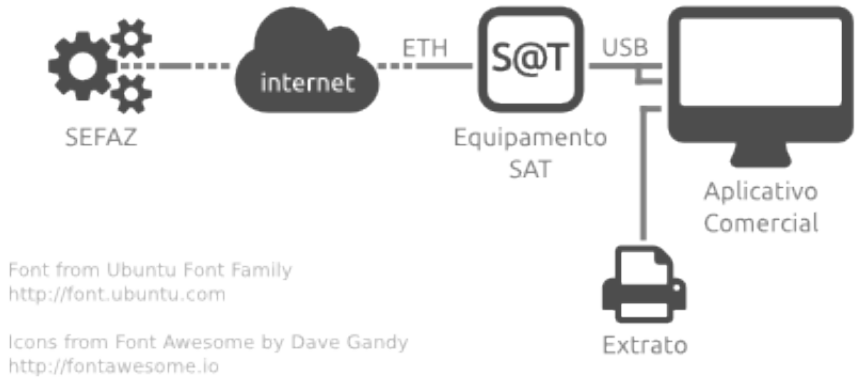
Nota: This documentation is a work in progress

Nota: Esta documentação é um trabalho em andamento

A Secretaria da Fazenda do Estado de São Paulo, SEFAZ/SP desenvolveu o projeto conhecido como **SAT-CF-e**, Sistema de Autorização e Transmissão de Cupons Fiscais eletrônicos, conforme descrito na Portaria CAT 147 de 05 de novembro de 2012. Na prática esse sistema torna obsoletos os Emissores de Cupons Fiscais (ECF) no Estado de São Paulo. Visite a página da Secretaria da Fazenda de São Paulo para outras informações.

Esta documentação diz respeito ao projeto SATCFe desenvolvido pela Base4 Sistemas Ltda 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 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 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 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 for capaz de 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.

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 configurar o *código de ativação* e o *número do caixa* de modo que esses dados não precisarão ser informados sempre que uma função for invocada:

```
from satcfe import conf

conf.codigo_ativacao = '123123123'
conf.numero_caixa = 1
```

No segundo cenário, basta configurar ao acesso ao SATHub:

```
from satcfe import conf

conf.numero_caixa = 15
conf.sathub.host = '192.168.0.101'
conf.sathub.port = 8088
```

A maneira como essas configurações serão persistidas pela aplicação comercial e como elas serão atribuídas na inicialização 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 SO (*shared object*, `.so`), de modo que é necessário indicar o caminho completo para a biblioteca e a convenção de chamada:

```
from satcomum import constantes
from satcfe import DLLSAT
from satcfe import ClienteSATLocal

cliente = ClienteSATLocal(DLLSAT(
        caminho='caminho/para/sat.dll',
        convencao=constantes.WINDOWS_STDCALL))

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
from satcfe import conf

conf.sathub.numero_caixa = 7
conf.sathub.host = '192.168.0.101'
conf.sathub.port = 8088

cliente = ClienteSATHub()

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(
        DLLSAT(caminho='caminho/para/sat.dll',
               convencao=constantes.WINDOWS_STDCALL),
        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. Porém, trata-se de uma aplicação baseada em Flask que é bastante conhecido e relativamente fácil de instalar.

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

>>> 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:
    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 porque, 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) 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
u'SAT em Opera\xe7\xe3o'
```

ConsultarStatusOperacional

A função `ConsultarStatusOperacional` (ER item 6.1.7) 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
u'Resposta com Sucesso'
```

```
>>> resp.NSERIE
320008889

>>> resp.STATUS_LAN
u'CONECTADO'

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

ConsultarNumeroSessao

A função `ConsultarNumeroSessao` (ER item 6.1.8) 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.

ExtrairLogs

A função `ExtrairLogs` (ER item 6.1.12) 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
u'Transfer\xeancia completa'

>>> resp.arquivoLog
u'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
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) 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()
u'-----BEGIN CERTIFICATE REQUEST-----
MIIBnTCCAQYCAQAwXTELMAkGA1UEBhMCU0cxETAPBgNVBAoTCE0yQ3J5cHRvMRIw
 ...
9rsQkRc9Urv9mRBIsredGnYECNeRaK5R1yzpOowninXC
-----END CERTIFICATE REQUEST-----
```

ComunicarCertificadoICPBRASIL

A função `ComunicarCertificadoICPBRASIL` (ER item 6.1.2) 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
u'Certificado transmitido com sucesso'
```

ConfigurarInterfaceDeRede

A função `ConfigurarInterfaceDeRede` (ER item 6.1.9) 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
u'Rede configurada com sucesso'
```

AssociarAssinatura

Por fazer

Escrever este tópico.

AtualizarSoftwareSAT

Por fazer

Escrever este tópico.

BloquearSAT

Por fazer

Escrever este tópico.

DesbloquearSAT

Por fazer

Escrever este tópico.

TrocarCodigoDeAtivacao

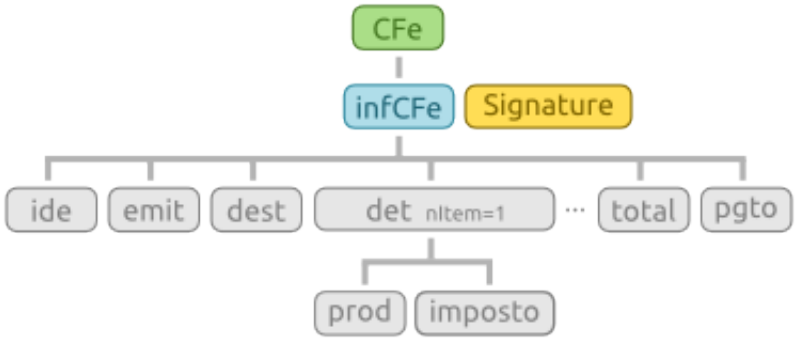
Por fazer

Escrever este tópico.

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	Ocorrência
AC		-	CFe	TAG raiz do CF-e	G	-		1-1
A - Dados do Cupom Fiscal Eletrônico								
Origem	#	ID	Campo	Descrição	Elemento	Pai	Tipo	Ocorrência
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 fiel
```

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>Catano
```

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 simples 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',
                                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')
```

3.4.4 Criando um CF-e de Cancelamento

Por fazer

Escrever este tópico.

3.5 Exemplos de Documentos

Abaixo estão relacionados alguns exemplos de documentos XML.

3.5.1 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 funcao-enviardadosvenda ou funcao-testefimafim.

```
<?xml version="1.0"?>
<CFe>
  <infCFe versaoDadosEnt="0.06">
    <ide>
      <CNPJ>08427847000169</CNPJ>
      <signAC>SGR-SAT SISTEMA DE GESTAO E RETAGUARDA DO SAT</signAC>
      <numeroCaixa>002</numeroCaixa>
    </ide>
    <emit>
      <CNPJ>61099008000141</CNPJ>
      <IE>111111111111</IE>
      <IM>12345</IM>
      <cRegTribISSQN>3</cRegTribISSQN>
      <indRatISSQN>N</indRatISSQN>
    </emit>
    <dest>
      <CPF>11122233396</CPF>
      <xNome>Joao de Teste</xNome>
    </dest>
    <entrega>
      <xLgr>Rua Armando Gulim</xLgr>
      <nro>65</nro>
      <xBairro>Parque Gloria III</xBairro>
      <xMun>Catanduva</xMun>
```

```
      <UF>SP</UF>
    </entrega>
    <det nItem="1">
      <prod>
        <cProd>123456</cProd>
        <xProd>BORRACHA STAEDTLER pvc-free</xProd>
        <CFOP>5102</CFOP>
        <uCom>UN</uCom>
        <qCom>1.0000</qCom>
        <vUnCom>5.75</vUnCom>
        <indRegra>A</indRegra>
      </prod>
      <imposto>
        <ICMS>
          <ICMSSN102>
            <Orig>2</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>10.00</vMP>
      </MP>
    </pgto>
  </infCFe>
</CFe>
```

3.5.2 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.

```
<?xml version="1.0"?>
<CFe>
  <infCFe Id="CFe35150761099008000141599000026310000100500297" versao="0.06" versaoDadosEnt="0.06" ve
    <ide>
      <cUF>35</cUF>
      <cNF>050029</cNF>
      <mod>59</mod>
```

[1] Também são chamados de "kit SAT".

```
      <nserieSAT>900002631</nserieSAT>
      <nCFe>000010</nCFe>
      <dEmi>20150709</dEmi>
      <hEmi>172317</hEmi>
      <cDV>7</cDV>
      <tpAmb>2</tpAmb>
      <CNPJ>08427847000169</CNPJ>
      <signAC>SGR-SAT SISTEMA DE GESTAO E RETAGUARDA DO SAT</signAC>
      <assinaturaQRCODE>fMery4SK4Q69PiHNSRSwjMKloMUGA4D6+6cPERqpsuJlC1MmUnGWuZi2+zEURx46vEqgQIETlJGk
      <numeroCaixa>002</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>
      <CPF>11122233396</CPF>
      <xNome>Joao de Teste</xNome>
    </dest>
    <entrega>
      <xLgr>Rua Armando Gulim</xLgr>
      <nro>65</nro>
      <xBairro>Parque Gloria III</xBairro>
      <xMun>Catanduva</xMun>
      <UF>SP</UF>
    </entrega>
    <det nItem="1">
      <prod>
        <cProd>123456</cProd>
        <xProd>BORRACHA STAEDTLER pvc-free</xProd>
        <CFOP>5102</CFOP>
        <uCom>UN</uCom>
        <qCom>1.0000</qCom>
        <vUnCom>5.75</vUnCom>
        <vProd>5.75</vProd>
        <indRegra>A</indRegra>
        <vItem>5.75</vItem>
      </prod>
      <imposto>
        <ICMS>
          <ICMSSN102>
            <Orig>2</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>5.75</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>5.75</vCFe>
    </total>
    <pgto>
      <MP>
        <cMP>01</cMP>
        <vMP>10.00</vMP>
      </MP>
      <vTroco>4.25</vTroco>
    </pgto>
  </infCFe>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <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="#CFe35150761099008000141599000026310000100500297">
        <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>ZaM4D/tOxLMGRqPpgDp5iDJyc9tFRIrty+UYGIGlTe4=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>JYCByJND++mEdkaCHBjLj+NLcwO1dVOlBV2awUzXndHIpfffouqQyWQJFpwGSdXtrPq1cjXg0cLdJKs/
    <KeyInfo>
      <X509Data>
        <X509Certificate>MIIG7zCCBNegAwIBAgIQPYgMyeNxYCOCHVsMXFBvODANBgkqhkiG9w0BAQsFADBnMQswCQYDVQQ
      </X509Data>
    </KeyInfo>
  </Signature>
</CFe>
```

Nota

3.6 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.

3.6.1 Módulo satcfe.base

class satcfe.base.**DLLSAT**(*caminho=None*, *convencao=1*)

Configura a localização da DLL do equipamento SAT e mantém uma referência carregada para a ela, conforme a convenção de chamada.

caminho

Caminho completo, incluindo o nome do arquivo, para a biblioteca SAT (DLL ou *shared object*).

Levanta ValueError Se a biblioteca não existir no caminho indicado.

carregar()

Carrega (ou recarrega) a biblioteca SAT.

Levanta ValueError Se a convenção de chamada não for reconhecida.

convencao

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

Levanta ValueError Se a convenção de chamada não for reconhecida.

ref

Uma referência para a biblioteca SAT carregada.

class satcfe.base.**FuncoesSAT**(*dll=None*, *numerador_sessao=None*)

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()*

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 2.3.1.
- **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*)
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 a ser cancelado, prefixada com o literal CFe.
- **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*)
Função ConfigurarInterfaceDeRede conforme ER SAT, item 6.1.9. Configurção da interface de comunicação do equipamento SAT.

Parâmetros **configuracao** – Instância de *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 string

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

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*)
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** – Uma 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*)
Função `TesteFimAFim` conforme ER SAT, item 6.1.6. Teste de comunicação entre a AC, o equipamento SAT e a SEFAZ.

Parâmetros **dados_venda** – Uma 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.

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: Os argumentos da função `TrocarCodigoDeAtivacao` requerem que o novo código de ativação seja especificado duas vezes (dois argumentos com o mesmo conteúdo, como confirmação). Este método irá simplesmente informar duas vezes o argumento `novo_codigo_ativacao` na função SAT, mantendo a confirmação do código de ativação fora do escopo desta API.

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.

```
>>> numerador = NumeroSessaoMemoria(tamanho=5)
>>> n1 = numerador()
>>> 100000 <= n1 <= 999999
True
>>> n1 in numerador
True
>>> n2 = numerador()
>>> n3 = numerador()
>>> n4 = numerador()
>>> n5 = numerador()
>>> len(set([n1, n2, n3, n4, n5]))
5
>>> n6 = numerador()
>>> n1 in numerador
False
```

3.6.2 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*)
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*)
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*

desbloquear_sat()
Sobrepõe *desbloquear_sat()*.

Retorna Uma resposta SAT padrão.

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

enviar_dados_venda(*dados_venda*)
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*)
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*

3.6.3 Módulo satcfe.clientesathub

class satcfe.clientesathub.**ClienteSATHub**(*dll=None*, *numerador_sessao=None*)
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.

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*)
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*)
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*

desbloquear_sat()
Sobrepõe *desbloquear_sat()*.

Retorna Uma resposta SAT padrão.

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

enviar_dados_venda(*dados_venda*)
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*)
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*

3.6.4 Módulo satcfe.entidades

Representação das entidades que compõem o layout do CF-e.

A documentação oficial para os atributos que as classes de entidades referenciam, estão na Especificação Técnica de Requisitos (ER) do SAT, item 4.2.2, Layout do Arquivo de Venda (CF-e-SAT) que pode ser obtido no site oficial.

Nem todas as classes que representam os grupos de informações do CF-e possuem o mesmo nome usado no item 4.2.2 do layout do arquivo de venda ou no item 4.2.3 do layout do arquivo de cancelamento. **Entretanto, todos os elementos e atributos, possuem exatamente o mesmo nome usado na ER SAT**.

A tabela abaixo, relaciona as classes de entidades com os grupos que elas representam:

Grupo	ID	Classe de Entidade
emit	C01	*Emitente*
dest	E01	*Destinatario*
entrega	G01	*LocalEntrega*
det	H01	*Detalhamento*
prod	I01	*ProdutoServico*
obsFiscoDet	I17	*ObsFiscoDet*
ICMS00	N02	*ICMS00*
ICMS40	N03	*ICMS40*
ICMSSN102	N04	*ICMSSN102*
ICMSSN900	N05	*ICMSSN900*
PISAliq	Q02	*PISAliq*
PISQtde	Q03	*PISQtde*
PISNT	Q04	*PISNT*
PISSN	Q05	*PISSN*
PISOutr	Q06	*PISOutr*
PISST	R01	*PISST*
COFINSAliq	S02	*COFINSAliq*
COFINSQtde	S03	*COFINSQtde*
COFINSNT	S04	*COFINSNT*
COFINSSN	S05	*COFINSSN*
COFINSOutr	S06	*COFINSOutr*
COFINSST	T01	*COFINSST*
ISSQN	U01	*ISSQN*
imposto	M01	*Imposto*
DescAcrEntr	W19	*DescAcrEntr*
MP	WA02	*MeioPagamento*
infAdic	Z01	*InformacoesAdicionais*

Hierarquia dos elementos XML do layout do CF-e, ER SAT, item 4.2.2.

Destinatário e Local de entrega:

```
infCFe (A01, 1)
|
+--- dest (E01, 1)
|
+--- entrega (G01, 0..1)
:
```

Detalhamento dos produtos/serviços e impostos:

```
infCFe (A01, 1)
|
+--- det (H01, 1..500)
|    |
|    +--- prod (I01, 1)
|    |    |
|    |    +--- obsFiscoDet (I17, 0..10)
|    |
|    +--- imposto (M01, 1)
|         |
|         +--- ICMS (N01, 0..1)
:         |    |
:         |    +--- ICMS00 (N02, 0..1)
.         |    |         > ICMS 00, 20, 90
.         |    |
          |    +--- ICMS40 (N03, 0..1)
          |    |         > ICMS 40, 41, 50, 60
          |    |
          |    +--- ICMSSN102 (N04, 0..1)
          |    |         > cRegTrib = 1, Simples Nacional
          |    |         > CSOSN 102, 300, 500
          |    |
          |    +--- ICMSSN900 (N05, 0..1)
          |              > cRegTrib = 1, Simples Nacional
          |              > CSOSN 900
          |
          +--- PIS (Q01, 1..1)
          |    |
          |    +--- PISAliq (Q02, 0..1)
          |    |         > CST 01, 02, 05
          |    |
          |    +--- PISQtde (Q03, 0..1)
          |    |         > CST 03
          |    |
          |    +--- PISNT (Q04, 0..1)
          |    |         > Não Tributado
          |    |         > CST 04, 06, 07, 08, 09
          |    |
          |    +--- PISSN (Q05, 0..1)
          |    |         > Simples Nacional
          |    |         > CST 49
          |    |
          |    +--- PISOutr (Q06, 0..1)
          |              > Outras Operações
          |              > CST 99
          |
          +--- PISST (R01, 0..1)
```

```
|          > Substituição Tributária
|
+--- COFINS (S01, 1..1)
|    |
|    +--- COFINSAliq (S02, 0..1)
|    |         > CST 01, 02, 05
|    |
|    +--- COFINSQtde (S03, 0..1)
|    |         > CST 03
|    |
|    +--- COFINSNT (S04, 0..1)
|    |         > Não Tributado
|    |         > CST 04, 06, 07, 08, 09
|    |
|    +--- COFINSSN (S05, 0..1)
|    |         > Simples Nacional
|    |         > CST 49
|    +--- COFINSOutr (S06, 0..1)
|              > Outras Operações
|              > CST 99
|
+--- COFINSST (T01, 0..1)
|          > Substituição Tributária
|
+--- ISSQN (U01, 0..1)
```

Totais:

```
infCFe (A01, 1)
|
+--- total (W01, 1)
|    |
|    +--- ICMSTot (W02, 0..1)
|    |          > Neste grupo, todos os elementos
:    |          > são calculados pelo equipamento SAT
:    |
.    +--- ISSQNTot (W12, 0..1)
.    |          > Neste grupo, todos os elementos
     |          > são calculados pelo equipamento SAT
     |
     +--- DescAcrEntr (W19, 0..1)
```

Pagamento:

```
infCFe (A01, 1)
|
+--- pgto (WA01, 1)
:    |
     +--- MP (WA02, 1..10)
```

class satcfe.entidades.**CFeCancelamento**(*destinatario=None*, ***kwargs*)

Representa um CF-e de cancelamento.

Parâmetros

- **destinatario** (Destinatario) –
- **chCanc** (*str*) –
- **CNPJ** (*str*) –

- **signAC** (*str*) –
- **numeroCaixa** (*int*) –

```
>>> cfecanc = CFeCancelamento(
...         chCanc='CFe01234567890123456789012345678901234567890123',
...         CNPJ='08427847000169',
...         signAC=constantes.ASSINATURA_AC_TESTE,
...         numeroCaixa=1)
>>> ET.tostring(cfecanc._xml())
'<CFeCanc><infCFe chCanc="CFe01234567890123456789012345678901234567890123"><ide><CNPJ>0842784700
```

destinatario
O *Destinatario* ou None.

class satcfe.entidades.**CFeVenda**(*emitente=None*, *destinatario=None*, *entrega=None*, *detalhamentos=[]*, *descontos_acrescimos_subtotal=None*, *pagamentos=[]*, *informacoes_adicionais=None*, ***kwargs*)

Representa um CF-e de venda.

Parâmetros

- **emitente** (Emitente) –
- **destinatario** (Destinatario) – *Opcional*
- **entrega** (LocalEntrega) – *Opcional*
- **detalhamentos** (*list*) –
- **descontos_acrescimos_subtotal** (DescAcrEntr) – *Opcional*
- **pagamentos** (*list*) –
- **informacoes_adicionais** (InformacoesAdicionais) – *Opcional*
- **versaoDadosEnt** (*str*) – *Opcional*
- **CNPJ** (*str*) –
- **signAC** (*str*) –
- **numeroCaixa** (*int*) –

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

```
>>> cfe = CFeVenda(
...         CNPJ='08427847000169',
...         signAC=constantes.ASSINATURA_AC_TESTE,
...         numeroCaixa=1,
...         emitente=Emitente(
...                 CNPJ='61099008000141',
...                 IE='111111111111',
...                 IM='12345',
...                 cRegTribISSQN=constantes.C15_SOCIEDADE_PROFISSIONAIS,
...                 indRatISSQN=constantes.C16_NAO_RATEADO))
>>> ET.tostring(cfe._xml())
'<CFe><infCFe versaoDadosEnt="0.06"><ide><CNPJ>08427847000169</CNPJ><signAC>SGR-SAT SISTEMA DE G
```

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*) –

```
>>> cofins = COFINSAliq(CST='01', vBC=Decimal('1.00'), pCOFINS=Decimal('0.0065'))
>>> ET.tostring(cofins._xml())
'<COFINSAliq><CST>01</CST><vBC>1.00</vBC><pCOFINS>0.0065</pCOFINS></COFINSAliq>'
```

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

Parâmetros **CST** (*str*) –

```
>>> cofins = COFINSNT(CST='04')
>>> ET.tostring(cofins._xml())
'<COFINSNT><CST>04</CST></COFINSNT>'
```

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.

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

```
>>> cofins = COFINSOutr(CST='99', vBC=Decimal('1.00'), pCOFINS=Decimal('0.0065'))
>>> ET.tostring(cofins._xml())
'<COFINSOutr><CST>99</CST><vBC>1.00</vBC><pCOFINS>0.0065</pCOFINS></COFINSOutr>'
>>> cofins = COFINSOutr(CST='99', qBCProd=Decimal('100.0000'), vAliqProd=Decimal('0.6500'))
>>> ET.tostring(cofins._xml())
'<COFINSOutr><CST>99</CST><qBCProd>100.0000</qBCProd><vAliqProd>0.6500</vAliqProd></COFINSOutr>'
```

```
    # atributo vBC depende de pCOFINS que não foi informado
    >>> cofins = COFINSOutr(CST='99', vBC=Decimal('1.00')) # vBC depende de pCOFINS
    >>> cofins._xml()
    Traceback (most recent call last):
     ...
    ValidationError: ...

    # atributo qBCProd depende de vAliqProd que não foi informado
    >>> cofins = COFINSOutr(CST='99', qBCProd=Decimal('100.0000'))
    >>> cofins._xml()
    Traceback (most recent call last):
     ...
    ValidationError: ...

    # neste caso, deve falhar pois vBC ou qBCProd não foram informados
    >>> cofins = COFINSOutr(CST='99')
    >>> cofins._xml()
    Traceback (most recent call last):
     ...
    ValidationError: Grupo 'COFINSOutr' requer exclusivamente 'vBC' ou 'qBCProd' (nenhum informado)

    # neste caso, deve falhar pois apenas um ou outro grupo pode ser informado:
    # ou informa-se vBC e pCOFINS ou informa-se qBCProd e vAliqProd
    >>> cofins = COFINSOutr(CST='99', vBC=Decimal('1.00'), pCOFINS=Decimal('1.00'), qBCProd=Decimal(
    >>> cofins._xml()
    Traceback (most recent call last):
     ...
    ValidationError: Grupo 'COFINSOutr' requer exclusivamente 'vBC' ou 'qBCProd' (ambos informados)

    # neste caso as falhara pela ausencia das dependencias:
    # pCOFINS depende de vBC e vAliqProd depende de qBCProd
    >>> cofins = COFINSOutr(CST='99', pCOFINS=Decimal('1.00'), vAliqProd=Decimal('1.00'))
    >>> cofins._xml()
    Traceback (most recent call last):
     ...
    ValidationError: ...
```

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

Grupo de COFINS tributado por quantidade, CST 03 (COFINSQtde, grupo S03).

Parâmetros

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

```
    >>> cofins = COFINSQtde(CST='03', qBCProd=Decimal('100.0000'), vAliqProd=Decimal('0.6500'))
    >>> ET.tostring(cofins._xml())
    '<COFINSQtde><CST>03</CST><qBCProd>100.0000</qBCProd><vAliqProd>0.6500</vAliqProd></COFINSQtde>'
```

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

Grupo de COFINS para contribuíntes do Simples Nacional, CST 49 (COFINSSN, grupo S05).

Parâmetros **CST** (*str*) –

```
    >>> cofins = COFINSSN(CST='49')
    >>> ET.tostring(cofins._xml())
    '<COFINSSN><CST>49</CST></COFINSSN>'
```

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.

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

```
>>> cofins = COFINSST(vBC=Decimal('1.00'), pCOFINS=Decimal('0.0065'))
>>> ET.tostring(cofins._xml())
'<COFINSST><vBC>1.00</vBC><pCOFINS>0.0065</pCOFINS></COFINSST>'
>>> cofins = COFINSST(qBCProd=Decimal('100.0000'), vAliqProd=Decimal('0.6500'))
>>> ET.tostring(cofins._xml())
'<COFINSST><qBCProd>100.0000</qBCProd><vAliqProd>0.6500</vAliqProd></COFINSST>'

# atributo vBC depende de pCOFINS que não foi informado
>>> cofins = COFINSST(vBC=Decimal('1.00')) # vBC depende de pCOFINS
>>> cofins._xml()
Traceback (most recent call last):
 ...
ValidationError: ...

# atributo qBCProd depende de vAliqProd que não foi informado
>>> cofins = COFINSST(qBCProd=Decimal('100.0000'))
>>> cofins._xml()
Traceback (most recent call last):
 ...
ValidationError: ...

# neste caso, deve falhar pois vBC ou qBCProd não foram informados
>>> cofins = COFINSST()
>>> cofins._xml()
Traceback (most recent call last):
 ...
ValidationError: Grupo 'COFINSST' requer exclusivamente 'vBC' ou 'qBCProd' (nenhum informado)

# neste caso, deve falhar pois apenas um ou outro grupo pode ser informado:
# ou informa-se vBC e pCOFINS ou informa-se qBCProd e vAliqProd
>>> cofins = COFINSST(vBC=Decimal('1.00'), pCOFINS=Decimal('1.00'), qBCProd=Decimal('1.00'), vAl
>>> cofins._xml()
Traceback (most recent call last):
 ...
ValidationError: Grupo 'COFINSST' requer exclusivamente 'vBC' ou 'qBCProd' (ambos informados)

# neste caso as falhara pela ausencia das dependencias:
# pCOFINS depende de vBC e vAliqProd depende de qBCProd
>>> cofins = COFINSST(pCOFINS=Decimal('1.00'), vAliqProd=Decimal('1.00'))
>>> cofins._xml()
Traceback (most recent call last):
 ...
ValidationError: ...
```

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

Parâmetros

- **vDescSubtot** (*Decimal*) –
- **vAcresSubtot** (*Decimal*) –
- **vCFeLei12741** (*Decimal*) –

```
>>> grupo = DescAcrEntr()
>>> ET.tostring(grupo._xml())
'<DescAcrEntr />'

>>> grupo = DescAcrEntr(
...          vDescSubtot=Decimal('0.01'),
...          vAcresSubtot=Decimal('0.02'),
...          vCFeLei12741=Decimal('0.03'))
>>> ET.tostring(grupo._xml())
'<DescAcrEntr><vDescSubtot>0.01</vDescSubtot><vAcresSubtot>0.02</vAcresSubtot><vCFeLei12741>0.03
```

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.

Note que os parâmetros CNPJ e CPF são mutuamente exclusivos.

```
>>> dest = Destinatario()
>>> ET.tostring(dest._xml(), encoding='utf-8')
'<dest />'
>>> dest = Destinatario(CNPJ='08427847000169')
>>> ET.tostring(dest._xml(), encoding='utf-8')
'<dest><CNPJ>08427847000169</CNPJ></dest>'
>>> dest = Destinatario(CPF='11122233396', xNome=u'Fulano Beltrano')
>>> ET.tostring(dest._xml(), encoding='utf-8')
'<dest><CPF>11122233396</CPF><xNome>Fulano Beltrano</xNome></dest>'
>>> dest = Destinatario(CPF='11122233396', CNPJ='08427847000169')
>>> dest._xml()
Traceback (most recent call last):
 ...
ValidationError: ...

# testa criação do XML para cancelamento; o nome deverá ser ignorado
>>> dest = Destinatario(CPF='11122233396', xNome=u'Fulano Beltrano')
>>> ET.tostring(dest._xml(cancelamento=True), encoding='utf-8')
'<dest><CPF>11122233396</CPF></dest>'
```

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

Parâmetros

- **produto** (ProdutoServico) –

- **imposto** (Imposto) –
- **infAdProd** (*str*) – *Opcional*

Note que o atributo XML `nItem` (`H02`) não é determinado aqui, mas atribuído automaticamente, conforme a sua posição na lista de *detalhamentos*.

```
>>> det = Detalhamento(
...         produto=ProdutoServico(
...                 cProd='123456',
...                 xProd='BORRACHA STAEDTLER',
...                 CFOP='5102',
...                 uCom='UN',
...                 qCom=Decimal('1.0000'),
...                 vUnCom=Decimal('5.75'),
...                 indRegra='A'),
...         imposto=Imposto(
...                 pis=PISSN(CST='49'),
...                 cofins=COFINSSN(CST='49'),
...                 icms=ICMSSN102(Orig='2', CSOSN='500')),
...         infAdProd='Teste')
>>> ET.tostring(det._xml(nItem=1))
'<det nItem="1"><prod><cProd>123456</cProd><xProd>BORRACHA STAEDTLER</xProd><CFOP>5102</CFOP><uC
```

imposto
O grupo de tributos incidentes no produto ou serviço ao qual o detalhamento se refere, como uma instância de *Imposto*.

produto
O produto ou serviço como uma instância de *ProdutoServico* ao qual o detalhamento se refere.

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`.

```
>>> emit = Emitente(CNPJ='08427847000169', IE='111222333444', indRatISSQN='S')
>>> ET.tostring(emit._xml())
'<emit><CNPJ>08427847000169</CNPJ><IE>111222333444</IE><indRatISSQN>S</indRatISSQN></emit>'

>>> emit = Emitente(
...         CNPJ='08427847000169',
...         IE='111222333444',
...         IM='123456789012345',
...         cRegTribISSQN='1',
...         indRatISSQN='S')
>>> ET.tostring(emit._xml())
'<emit><CNPJ>08427847000169</CNPJ><IE>111222333444</IE><IM>123456789012345</IM><cRegTribISSQN>1<
```

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 sobre-escrever a implementação do método `_construir_elemento_xml`, definir o atributo `_schema` e, quando necessário, implementar uma especialização do validador Cerberus 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.**ICMS00**(***kwargs*)

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

Parâmetros

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

```
>>> icms = ICMS00(Orig='0', CST='00', pICMS=Decimal('18.00'))
>>> ET.tostring(icms._xml())
'<ICMS00><Orig>0</Orig><CST>00</CST><pICMS>18.00</pICMS></ICMS00>'
```

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

Grupo de tributação do ICMS 40, 41, 50 e 60 (`ICMS40`, grupo `N03`).

Parâmetros

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

```
>>> icms = ICMS40(Orig='0', CST='60')
>>> ET.tostring(icms._xml())
'<ICMS40><Orig>0</Orig><CST>60</CST></ICMS40>'
```

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

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

Parâmetros

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

```
>>> icms = ICMSSN102(Orig='0', CSOSN='500')
>>> ET.tostring(icms._xml())
'<ICMSSN102><Orig>0</Orig><CSOSN>500</CSOSN></ICMSSN102>'
```

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

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

Parâmetros

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

```
>>> icms = ICMSSN900(Orig='0', CSOSN='900', pICMS=Decimal('18.00'))
>>> ET.tostring(icms._xml())
'<ICMSSN900><Orig>0</Orig><CSOSN>900</CSOSN><pICMS>18.00</pICMS></ICMSSN900>'
```

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*) –

```
>>> issqn = ISSQN(vDeducISSQN=Decimal('10.00'), vAliq=Decimal('7.00'), cNatOp='01', indIncFisc='
>>> ET.tostring(issqn._xml())
'<ISSQN><vDeducISSQN>10.00</vDeducISSQN><vAliq>7.00</vAliq><cNatOp>01</cNatOp><indIncFisc>2</ind

>>> issqn = ISSQN(vDeducISSQN=Decimal('10.00'), vAliq=Decimal('7.00'), cNatOp='01', indIncFisc='
>>> ET.tostring(issqn._xml())
'<ISSQN><vDeducISSQN>10.00</vDeducISSQN><vAliq>7.00</vAliq><cMunFG>3511102</cMunFG><cListServ>01
```

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** (*str*) – Deve ser uma instância de uma dlas classes dos grupos de COFINS (*COFINSAliq*, *COFINSQtde*, *COFINSNT*, *COFINSSN* ou *COFINSOutr*).
- **cofinsst** (*str*) – *Opcional* Instância de *COFINSST* ou None.
- **issqn** (*str*) – *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.

```
>>> imposto = Imposto(
...          vItem12741=Decimal('0.10'),
...          icms=ICMS00(Orig='0', CST='00', pICMS=Decimal('18.00')),
...          pis=PISSN(CST='49'),
...          cofins=COFINSSN(CST='49'))
>>> ET.tostring(imposto._xml())
'<imposto><vItem12741>0.10</vItem12741><ICMS><ICMS00><Orig>0</Orig><CST>00</CST><pICMS>18.00</pI
```

```
    # sem pis
    >>> imposto = Imposto(cofins=COFINSSN(CST='49'))
    >>> imposto._xml()
    Traceback (most recent call last):
     ...
    ValidationError: 'Imposto' (grupo M01 'imposto') atributo 'pis' nao pode ser 'None'

    # sem cofins
    >>> imposto = Imposto(pis=PISSN(CST='49'))
    >>> imposto._xml()
    Traceback (most recent call last):
     ...
    ValidationError: 'Imposto' (grupo M01 'imposto') atributo 'cofins' nao pode ser 'None'
```

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*

```
    >>> grupo = InformacoesAdicionais()
    >>> ET.tostring(grupo._xml())
    '<infAdic />'

    >>> grupo = InformacoesAdicionais(infCpl='Teste')
    >>> ET.tostring(grupo._xml())
    '<infAdic><infCpl>Teste</infCpl></infAdic>'
```

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*) –

```
>>> entrega = LocalEntrega()
>>> ET.tostring(entrega._xml(), encoding='utf-8')
Traceback (most recent call last):
 ...
ValidationError: ...
>>> entrega.xLgr = 'Rua Armando Gulim'
>>> entrega.nro = '65'
>>> entrega.xBairro = 'Parque Gloria III'
>>> entrega.xMun = 'Catanduva'
>>> entrega.UF = 'SP'
>>> ET.tostring(entrega._xml(), encoding='utf-8')
'<entrega><xLgr>Rua Armando Gulim</xLgr><nro>65</nro><xBairro>Parque Gloria III</xBairro><xMun>C
```

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

Parâmetros

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

```
>>> mp = MeioPagamento(cMP='01', vMP=Decimal('10.00'))
>>> ET.tostring(mp._xml())
'<MP><cMP>01</cMP><vMP>10.00</vMP></MP>'

>>> mp = MeioPagamento(cMP='01', vMP=Decimal('10.00'), cAdmC='999')
>>> ET.tostring(mp._xml())
'<MP><cMP>01</cMP><vMP>10.00</vMP><cAdmC>999</cAdmC></MP>'
```

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

Parâmetros

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

```
>>> obs = ObsFiscoDet(xCampoDet='Cod. Produto ANP', xTextoDet='320101001')
>>> ET.tostring(obs._xml())
'<obsFiscoDet xCampoDet="Cod. Produto ANP"><xTextoDet>320101001</xTextoDet></obsFiscoDet>'
```

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*) –

```
>>> pis = PISAliq(CST='01', vBC=Decimal('1.00'), pPIS=Decimal('0.0065'))
>>> ET.tostring(pis._xml())
'<PISAliq><CST>01</CST><vBC>1.00</vBC><pPIS>0.0065</pPIS></PISAliq>'
```

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

Parâmetros **CST** (*str*) –

```
>>> pis = PISNT(CST='04')
>>> ET.tostring(pis._xml())
'<PISNT><CST>04</CST></PISNT>'
```

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.

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

```
>>> pis = PISOutr(CST='99', vBC=Decimal('1.00'), pPIS=Decimal('0.0065'))
>>> ET.tostring(pis._xml())
'<PISOutr><CST>99</CST><vBC>1.00</vBC><pPIS>0.0065</pPIS></PISOutr>'
>>> pis = PISOutr(CST='99', qBCProd=Decimal('100.0000'), vAliqProd=Decimal('0.6500'))
>>> ET.tostring(pis._xml())
'<PISOutr><CST>99</CST><qBCProd>100.0000</qBCProd><vAliqProd>0.6500</vAliqProd></PISOutr>'

# atributo vBC depende de pPIS que não foi informado
>>> pis = PISOutr(CST='99', vBC=Decimal('1.00')) # vBC depende de pPIS
>>> pis._xml()
Traceback (most recent call last):
 ...
ValidationError: ...

# atributo qBCProd depende de vAliqProd que não foi informado
>>> pis = PISOutr(CST='99', qBCProd=Decimal('100.0000'))
>>> pis._xml()
Traceback (most recent call last):
 ...
ValidationError: ...

# neste caso, deve falhar pois vBC ou qBCProd não foram informados
>>> pis = PISOutr(CST='99')
>>> pis._xml()
Traceback (most recent call last):
 ...
ValidationError: Grupo 'PISOutr' requer exclusivamente 'vBC' ou 'qBCProd' (nenhum informado)

# neste caso, deve falhar pois apenas um ou outro grupo pode ser informado:
# ou informa-se vBC e pPIS ou informa-se qBCProd e vAliqProd
>>> pis = PISOutr(CST='99', vBC=Decimal('1.00'), pPIS=Decimal('1.00'), qBCProd=Decimal('1.00'),
>>> pis._xml()
Traceback (most recent call last):
 ...
ValidationError: Grupo 'PISOutr' requer exclusivamente 'vBC' ou 'qBCProd' (ambos informados)
```

```
    # neste caso as falhara pela ausencia das dependencias:
    # pPIS depende de vBC e vAliqProd depende de qBCProd
    >>> pis = PISOutr(CST='99', pPIS=Decimal('1.00'), vAliqProd=Decimal('1.00'))
    >>> pis._xml()
    Traceback (most recent call last):
     ...
    ValidationError: ...
```

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

Parâmetros

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

```
    >>> pis = PISQtde(CST='03', qBCProd=Decimal('100.0000'), vAliqProd=Decimal('0.6500'))
    >>> ET.tostring(pis._xml())
    '<PISQtde><CST>03</CST><qBCProd>100.0000</qBCProd><vAliqProd>0.6500</vAliqProd></PISQtde>'
```

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

Parâmetros **CST** (*str*) –

```
    >>> pis = PISSN(CST='49')
    >>> ET.tostring(pis._xml())
    '<PISSN><CST>49</CST></PISSN>'
```

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.

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

```
    >>> pis = PISST(vBC=Decimal('1.00'), pPIS=Decimal('0.0065'))
    >>> ET.tostring(pis._xml())
    '<PISST><vBC>1.00</vBC><pPIS>0.0065</pPIS></PISST>'
    >>> pis = PISST(qBCProd=Decimal('100.0000'), vAliqProd=Decimal('0.6500'))
    >>> ET.tostring(pis._xml())
    '<PISST><qBCProd>100.0000</qBCProd><vAliqProd>0.6500</vAliqProd></PISST>'

    # atributo vBC depende de pPIS que não foi informado
    >>> pis = PISST(vBC=Decimal('1.00')) # vBC depende de pPIS
    >>> pis._xml()
    Traceback (most recent call last):
     ...
    ValidationError: ...
```

```
    # atributo qBCProd depende de vAliqProd que não foi informado
    >>> pis = PISST(qBCProd=Decimal('100.0000'))
    >>> pis._xml()
    Traceback (most recent call last):
     ...
    ValidationError: ...

    # neste caso, deve falhar pois vBC ou qBCProd não foram informados
    >>> pis = PISST()
    >>> pis._xml()
    Traceback (most recent call last):
     ...
    ValidationError: Grupo 'PISST' requer exclusivamente 'vBC' ou 'qBCProd' (nenhum informado)

    # neste caso, deve falhar pois apenas um ou outro grupo pode ser informado:
    # ou informa-se vBC e pPIS ou informa-se qBCProd e vAliqProd
    >>> pis = PISST(vBC=Decimal('1.00'), pPIS=Decimal('1.00'), qBCProd=Decimal('1.00'), vAliqProd=De
    >>> pis._xml()
    Traceback (most recent call last):
     ...
    ValidationError: Grupo 'PISST' requer exclusivamente 'vBC' ou 'qBCProd' (ambos informados)

    # neste caso as falhara pela ausencia das dependencias:
    # pPIS depende de vBC e vAliqProd depende de qBCProd
    >>> pis = PISST(pPIS=Decimal('1.00'), vAliqProd=Decimal('1.00'))
    >>> pis._xml()
    Traceback (most recent call last):
     ...
    ValidationError: ...
```

class satcfe.entidades.**ProdutoServico**(*observacoes_fisco=[]*, ***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*

```
    # apenas os atributos requeridos;
    # note que, diferente da NF-e/NFC-e a ER SAT indica que o
    # atributo NCM não é obrigatório
    >>> prod = ProdutoServico(
```

```
...          cProd='123456',
...          xProd='BORRACHA STAEDTLER',
...          CFOP='5102',
...          uCom='UN',
...          qCom=Decimal('1.0000'),
...          vUnCom=Decimal('5.75'),
...          indRegra='A')
>>> ET.tostring(prod._xml())
'<prod><cProd>123456</cProd><xProd>BORRACHA STAEDTLER</xProd><CFOP>5102</CFOP><uCom>UN</uCom><qC

# todos os atributos (se vDesc for informado, então não informa vOutro)
>>> prod = ProdutoServico(
...          cProd='123456',
...          cEAN='4007817525074',
...          xProd='BORRACHA STAEDTLER',
...          NCM='40169200',
...          CFOP='5102',
...          uCom='UN',
...          qCom=Decimal('1.0000'),
...          vUnCom=Decimal('5.75'),
...          indRegra='A',
...          vDesc=Decimal('0.25'))
>>> ET.tostring(prod._xml())
'<prod><cProd>123456</cProd><cEAN>4007817525074</cEAN><xProd>BORRACHA STAEDTLER</xProd><NCM>4016

# todos os atributos (informando vOutro)
>>> prod = ProdutoServico(
...          cProd='123456',
...          cEAN='4007817525074',
...          xProd='BORRACHA STAEDTLER',
...          NCM='40169200',
...          CFOP='5102',
...          uCom='UN',
...          qCom=Decimal('1.0000'),
...          vUnCom=Decimal('5.75'),
...          indRegra='A',
...          vOutro=Decimal('0.25'))
>>> ET.tostring(prod._xml())
'<prod><cProd>123456</cProd><cEAN>4007817525074</cEAN><xProd>BORRACHA STAEDTLER</xProd><NCM>4016

# informa vDesc e vOutro, não deve validar
>>> prod = ProdutoServico(
...          cProd='123456',
...          xProd='BORRACHA STAEDTLER',
...          CFOP='5102',
...          uCom='UN',
...          qCom=Decimal('1.0000'),
...          vUnCom=Decimal('5.75'),
...          indRegra='A',
...          vDesc=Decimal('0.25'),
...          vOutro=Decimal('0.25'))
>>> prod._xml()
Traceback (most recent call last):
  ...
ValidationError: 'ProdutoServico' (grupo H01 'prod') atributos 'vDesc' e 'vOutro' sao mutuamente
```

observacoes_fisco

Cada produto, pode opcionalmente, conter uma lista de campos de uso livre do fisco, cujos campos e

valores são representados por instâncias da classe *ObsFiscoDet*.

3.6.5 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.

3.6.6 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 o método 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, se for o caso, 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.

```
>>> from satcomum import constantes
>>> conf = ConfiguracaoRede(
...          tipoInter=constantes.REDE_TIPOINTER_ETHE,
...          tipoLan=constantes.REDE_TIPOLAN_DHCP)
>>> ET.tostring(conf._xml())
'<config><tipoInter>ETHE</tipoInter><tipoLan>DHCP</tipoLan></config>'
```

3.6.7 Módulo `satcfe.util`

satcfe.util.**as_ascii**(*value*)

Converte a sequência unicode para `str` ou apenas retorna o argumento.

```
>>> type(as_ascii('testando'))
<type 'str'>
>>> type(as_ascii(u'bênção'))
<type 'str'>
>>> as_ascii(u'b\u00EAn\u00E7\u00E3o')
'bencao'
```

satcfe.util.**as_clean_unicode**(*value*)

Resulta na conversão do argumento para `unicode` e a subsequente remoção dos espaços em branco das bordas.

```
>>> as_clean_unicode('abc')
u'abc'
>>> as_clean_unicode('abc\n')
u'abc'
>>> as_clean_unicode(' \tabc \t \n   ')
u'abc'
```

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.

```
>>> import datetime
>>> as_date('20150709')
datetime.date(2015, 7, 9)
>>> as_date('20150709\n')
datetime.date(2015, 7, 9)
>>> as_date('  \n')
Traceback (most recent call last):
 ...
ValueError: ...
```

satcfe.util.**as_date_or_none**(*value*)

Converte uma sequência string para um objeto `datetime.date` ou resulta em `None` se a sequência estiver vazia após terem sido removidos espaços em branco das bordas.

```
>>> import datetime
>>> as_date_or_none('20150709')
datetime.date(2015, 7, 9)
>>> as_date_or_none('20150709\n')
datetime.date(2015, 7, 9)
>>> assert as_date_or_none('  \n') is 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.

```
>>> import datetime
>>> as_datetime('20150709143944')
datetime.datetime(2015, 7, 9, 14, 39, 44)
>>> as_datetime('20150709143944\n')
datetime.datetime(2015, 7, 9, 14, 39, 44)
>>> as_datetime(' \t \n ')
Traceback (most recent call last):
 ...
ValueError: ...
```

satcfe.util.**as_datetime_or_none**(*value*)
Converte uma sequência string para um objeto datetime.datetime ou resulta em None se a sequência estiver vazia após terem sido removidos espaços em branco das bordas.

```
>>> import datetime
>>> as_datetime_or_none('20150709143944')
datetime.datetime(2015, 7, 9, 14, 39, 44)
>>> as_datetime_or_none('20150709143944\n')
datetime.datetime(2015, 7, 9, 14, 39, 44)
>>> assert as_datetime_or_none(' \t \n ') is None
```

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

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.

```
>>> normalizar_ip('010.000.000.001')
'10.0.0.1'
>>> normalizar_ip('10.0.0.1')
'10.0.0.1'
>>> normalizar_ip('')
Traceback (most recent call last):
 ...
ValueError: invalid literal for int() with base 10: ''
```

3.6.8 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 (unicode)
mensagem (unicode)
cod (unicode)
mensagemSEFAZ (unicode)
```

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.

CAMPOS = ((‘numeroSessao’, <type ‘int’>), (‘EEEEE’, <type ‘unicode’>), (‘mensagem’, <type ‘unicode’>), (‘cod’, <type ‘u

Campos padrão esperados em uma resposta e a sua função de conversão para o tipo Python, a partir da resposta original (unicode).

static associar_assinatura(*retorno*)

Constrói uma *RespostaSAT* para o retorno (unicode) da função *associar_assinatura()*.

static atualizar_software_sat(*retorno*)

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

static bloquear_sat(*retorno*)

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

static comunicar_certificado_icpbrasil(*retorno*)

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

static configurar_interface_de_rede(*retorno*)

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

static consultar_sat(*retorno*)

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

static desbloquear_sat(*retorno*)

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

static trocar_codigo_de_ativacao(*retorno*)

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

satcfe.resposta.padrao.**analisar_retorno**(*retorno*, *classe_resposta=<class ‘satcfe.resposta.padrao.RespostaSAT’>*, *campos=((‘numeroSessao’, <type ‘int’>), (‘EEEEE’, <type ‘unicode’>), (‘mensagem’, <type ‘unicode’>), (‘cod’, <type ‘unicode’>), (‘mensagemSEFAZ’, <type ‘unicode’>))*, *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.

O campos devem ser especificados como uma tupla onde cada elemento da tupla deverá ser uma tupla contendo dois elementos: o nome do campo e uma função de conversão a partir de uma string unicode. Por exemplo:

```
>>> retorno = '123456|08000|SAT em operacao||'
>>> resposta = analisar_retorno(retorno, funcao='ConsultarSAT')
>>> resposta.numeroSessao
123456
>>> resposta.EEEEE
u'08000'
>>> resposta.mensagem
u'SAT em operacao'
>>> resposta.cod
u''
>>> resposta.mensagemSEFAZ
u''
>>> resposta.atributos.funcao
'ConsultarSAT'
>>> resposta.atributos.verbatim
'123456|08000|SAT em operacao||'
```

Parâmetros

- **retorno** (*unicode*) – O conteúdo **unicode** da resposta retornada pela função da DLL SAT.
- **classe_resposta** (*type*) – O tipo *RespostaSAT* ou especialização que irá representar o retorno, após sua decomposição em campos.
- **campos** (*tuple*) – Especificação dos campos (nomes) e seus conversores a a partir do tipo `unicode`.
- **campos_alternativos** (*list*) – Especifica conjuntos de campos alternativos que serão considerados caso o número de campos encontrados na resposta não coincida com o número de campos especificados em `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 (unicode)
mensagem (unicode)
cod (unicode)
mensagemSEFAZ (unicode)
CSR (unicode)
```

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

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

Parâmetros **retorno** (*unicode*) – 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 (unicode)
CCCC (unicode)
mensagem (unicode)
cod (unicode)
mensagemSEFAZ (unicode)
arquivoCFeBase64 (unicode)
timeStamp (datetime.datetime)
chaveConsulta (unicode)
valorTotalCFe (decimal.Decimal)
CPFCNPJValue (unicode)
assinaturaQRCODE (unicode)
```

Em caso de falha, são esperados apenas os atributos:

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

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

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

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

qrcode()
Resulta nos dados que compõem o QRCode.

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

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** (*unicode*) – Retorno da função ConsultarNumeroSessao.

Módulo satcfe.resposta.consultarstatusoperacional

satcfe.resposta.consultarstatusoperacional.**ESTADOS_OPERACAO = ((0, u'Desbloqueado'), (1, u'Bloqueado pe**
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	unicode
mensagem	unicode
cod	unicode
mensagemSEFAZ	unicode
NSERIE	unicode
TIPO_LAN	unicode
LAN_IP	str
LAN_MAC	unicode
LAN_MASK	str
LAN_GW	str
LAN_DNS_1	str
LAN_DNS_2	str
STATUS_LAN	unicode
NIVEL_BATERIA	unicode
MT_TOTAL	unicode
MT_USADA	unicode
DH_ATUAL	datetime.datetime
VER_SB	unicode
VER_LAYOUT	unicode
ULTIMO_CF_E_SAT	unicode
LISTA_INICIAL	unicode
LISTA_FINAL	unicode
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*.

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

Parâmetros **retorno** (*unicode*) – 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 (unicode)
CCCC (unicode)
mensagem (unicode)
cod (unicode)
mensagemSEFAZ (unicode)
arquivoCFeSAT (unicode)
timeStamp (datetime.datetime)
chaveConsulta (unicode)
valorTotalCFe (decimal.Decimal)
CPFCNPJValue (unicode)
assinaturaQRCODE (unicode)
```

Em caso de falha, são esperados apenas os atributos:

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

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

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

Parâmetros **retorno** (*unicode*) – Retorno da função EnviarDadosVenda.

qrcode()
Resulta nos dados que compõem o QRCode.

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

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 (unicode)
mensagem (unicode)
cod (unicode)
mensagemSEFAZ (unicode)
arquivoLog (unicode)
```

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

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

Parâmetros **retorno** (*unicode*) – Retorno da função `ExtrairLogs`.

conteudo()
Retorna o conteúdo do log decodificado.

salvar(*destino=None*, *prefix='tmp'*, *suffix='-sat.log'*)
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"`.

Retorna Retorna o caminho completo para o arquivo salvo.

Tipo de retorno str

Levanta IOError Se o destino for informado e o arquivo 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 (unicode)
mensagem (unicode)
cod (unicode)
mensagemSEFAZ (unicode)
arquivoCFeBase64 (unicode)
timeStamp (datetime.datetime)
numDocFiscal (int)
chaveConsulta (unicode)
```

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

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

Parâmetros **retorno** (*unicode*) – 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.

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

CAPÍTULO 4

Tabelas e Índices

- genindex
- modindex
- search

CAPÍTULO 5

Glossário

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.

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.

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.

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.

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.

S

A

B

C

D

E

F

G

I