AvaTax Ruby Software Development Kit

User Guide

Version 1.0.10

Revision date: 10/18/2013



Avalara may have patents, patent applications, trademarks, copyrights, or other intellectual property rights governing the subject matter in this document.  Except as expressly provided in any written license agreement from Avalara, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2013 Avalara, Inc.  All rights reserved.

Avalara, AvaTax Calc, AvaTax Returns, AvaTax Certs, AvaRates, TrustFile, BPObridge, Laserbridge+, Sales Taxll, Sales TaxPC, SalestaxPC+, StatetaxII, and StatetaxPC are either registered trademarks or trademarks of Avalara, Inc. in the United States or other countries.

All other trademarks are property of their respective owners.

Contents

Contents

[Introduction 3](#_Toc370378678)

[Installing your SDK Product 4](#_Toc370378679)

[Quick Start 5](#_Toc370378680)

[Process Overview 10](#_Toc370378681)

[How It All Works 11](#_Toc370378682)

[Collecting Information 11](#_Toc370378683)

[Processing Information 11](#_Toc370378684)

[Drop Shipping 12](#_Toc370378685)

[Integration Touch Points 14](#_Toc370378686)

[About SDK Sample Code 14](#_Toc370378687)

[Getting Started 14](#_Toc370378688)

[Ping 14](#_Toc370378689)

[GetTax 14](#_Toc370378690)

[PostTax and CommitTax 16](#_Toc370378691)

[NewDocCode Property 17](#_Toc370378692)

[Committing a Document—a “Two Stage Process” 17](#_Toc370378693)

[Stage One 18](#_Toc370378694)

[A success message should result and should be passed to the ERP to indicate that both the ERP and AvaTax databases are in sync. A failure or “NOT” success (Boolean) should generate aStage Two 18](#_Toc370378695)

[Failure Situations 18](#_Toc370378696)

[CancelTax 18](#_Toc370378697)

[GetTaxHistory 19](#_Toc370378698)

[Validate 19](#_Toc370378699)

[Required Elements, Limitations, and Security 21](#_Toc370378700)

[Profile.Client Property 21](#_Toc370378701)

[Number of Lines per Document 21](#_Toc370378702)

[Web Services Security 21](#_Toc370378703)

[AvaTax Web Service and Proxy Server Settings 21](#_Toc370378704)

[Connecting to the AvaTax Service and DNS Lookups 22](#_Toc370378705)

[Best Practices & Tips 23](#_Toc370378706)

[GetTax / PostTax Commit Method 23](#_Toc370378707)

[Posting and Committing Invoices (Documents) 23](#_Toc370378708)

[AvaTax Document States 24](#_Toc370378709)

[Detail Level Enumeration 24](#_Toc370378710)

[Handling Return Invoices 25](#_Toc370378711)

[Handling Shipping and Freight 27](#_Toc370378712)

[Handling Discounts (or Gift Certificates) 27](#_Toc370378713)

[Error Messages 28](#_Toc370378714)

[Address Validation 30](#_Toc370378715)

[Address Validation - GetTax Method 32](#_Toc370378716)

[Tax Compliance 34](#_Toc370378717)

[What is NEXUS? 34](#_Toc370378718)

[ItemCode vs. TaxCode and Taxability Rules 35](#_Toc370378719)

[Going Live (Production) 36](#_Toc370378720)

[Production Credentials 36](#_Toc370378721)

[Migration Services 36](#_Toc370378722)

[System Outages 36](#_Toc370378723)

[Frequently Asked Questions: 36](#_Toc370378724)

[Resources 38](#_Toc370378725)

[Avalara University Free Training 38](#_Toc370378726)

[Admin Console, Web Service, Customer Center Access 38](#_Toc370378727)

[Admin Console Login 38](#_Toc370378728)

[Web Service Access (for the connector API) 38](#_Toc370378729)

[Avalara Customer Center Login 39](#_Toc370378730)

[Product Download 39](#_Toc370378731)

[Resetting your License Key 40](#_Toc370378732)

[SDK Product Documentation 41](#_Toc370378733)

[APPENDIX A: (Required Fields) 42](#_Toc370378734)

[APPENDIX B: (Common System Messages) 43](#_Toc370378735)

[APPENDIX C: (Hybrid Model) 45](#_Toc370378736)

[APPENDIX D: Billable Transactions SDK 46](#_Toc370378737)

[APPENDIX E: SDK Analysis Check List 47](#_Toc370378738)

[Avalara Support 49](#_Toc370378739)

[How to Get Help with SDK Integration 49](#_Toc370378740)

[Index 50](#_Toc370378741)

# Introduction

Avalara provides automated sales tax solutions to streamline cumbersome, error-prone tax compliance processes and reduce the risk of loss or penalty in case of an audit. Our automated solutions automatically perform address validation, jurisdiction research and rate calculation and allow you to manage even the most complicated tax issues, such as situs[[1]](#footnote-1), nexus, tax tiers, tax holidays, exemptions, certificate management and product taxability rules.

AvaTax Calc is a powerful, online sales tax compliance Solution. Sales Tax SDK is designed to integrate **[](http://www.avalara.com/products/avatax/calc)**with virtually any application, whether it is written in Ruby, PHP, .NET, Java, or a COM-compliant language. Our powerful Software Development Kit (SDK), available for AvaTax Calc and AvaRates, allows you to easily integrate our sales and use tax solutions with virtually any financial, e-commerce, point-of-sale, custom-built or other third-party application

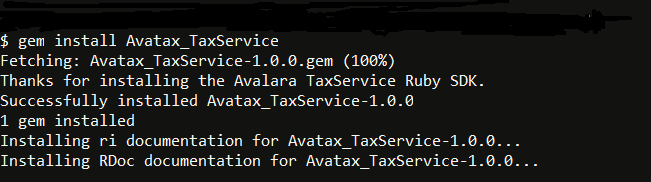
Avalara's Sales Tax SDK makes the AvaTax Calc and AvaRates web services available to developer partners providing full testing and support services as well as optional time-saving adapter samples. The Sales Tax SDK will make short, efficient work of your integration.

## Installing your SDK Product

The SDK’s for the Avatax Tax and Address services are provided as Ruby gems.

You can install via RubyGems.org ….

**gem install Avatax\_TaxService**



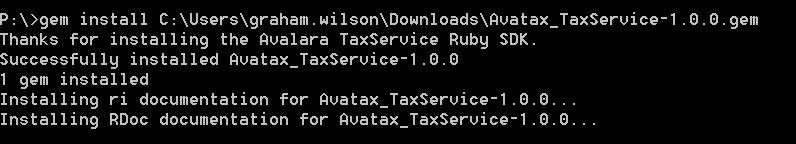
If you are intending to use our address validation service then install the Avatax\_AddressService gem also.

**gem install Avatax\_AddressService**

or you can download the gems from our developer web site …

<http://developer.avalara.com/api-docs/api-sample-code>

… and install from a command line similar to the example below.

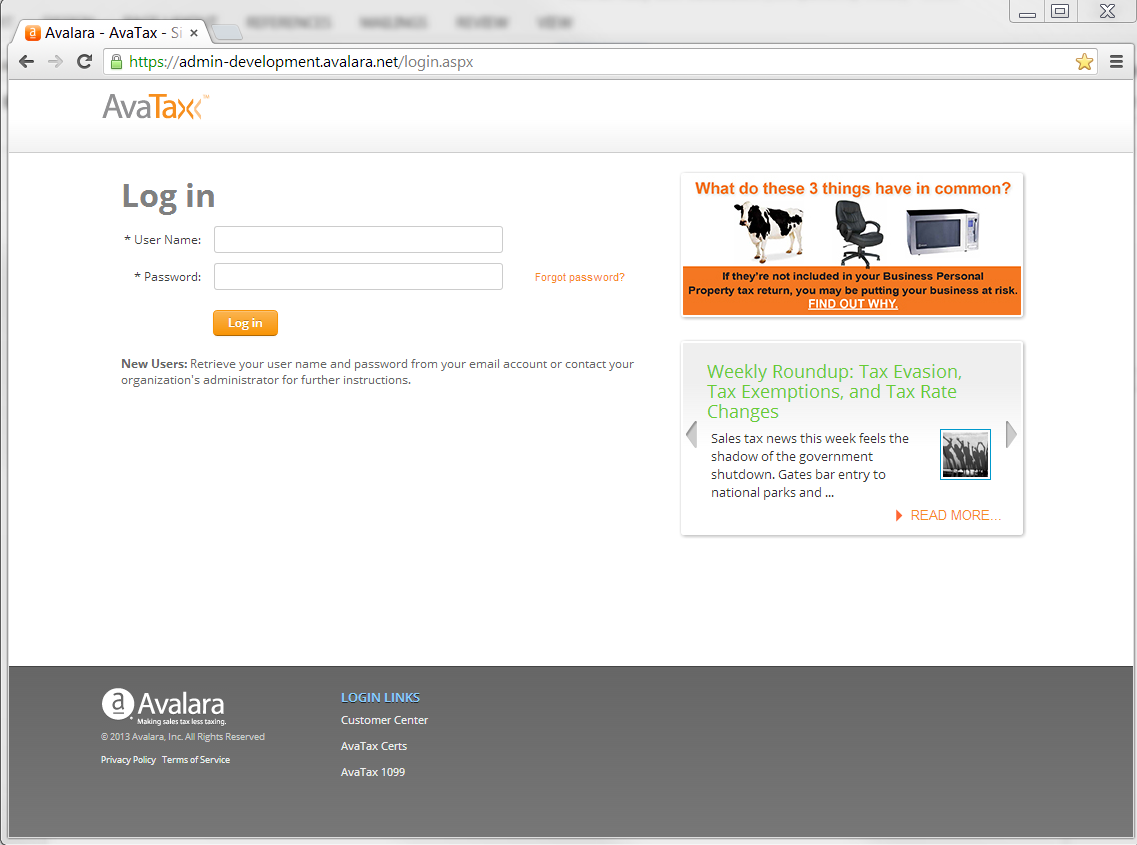


## Quick Start

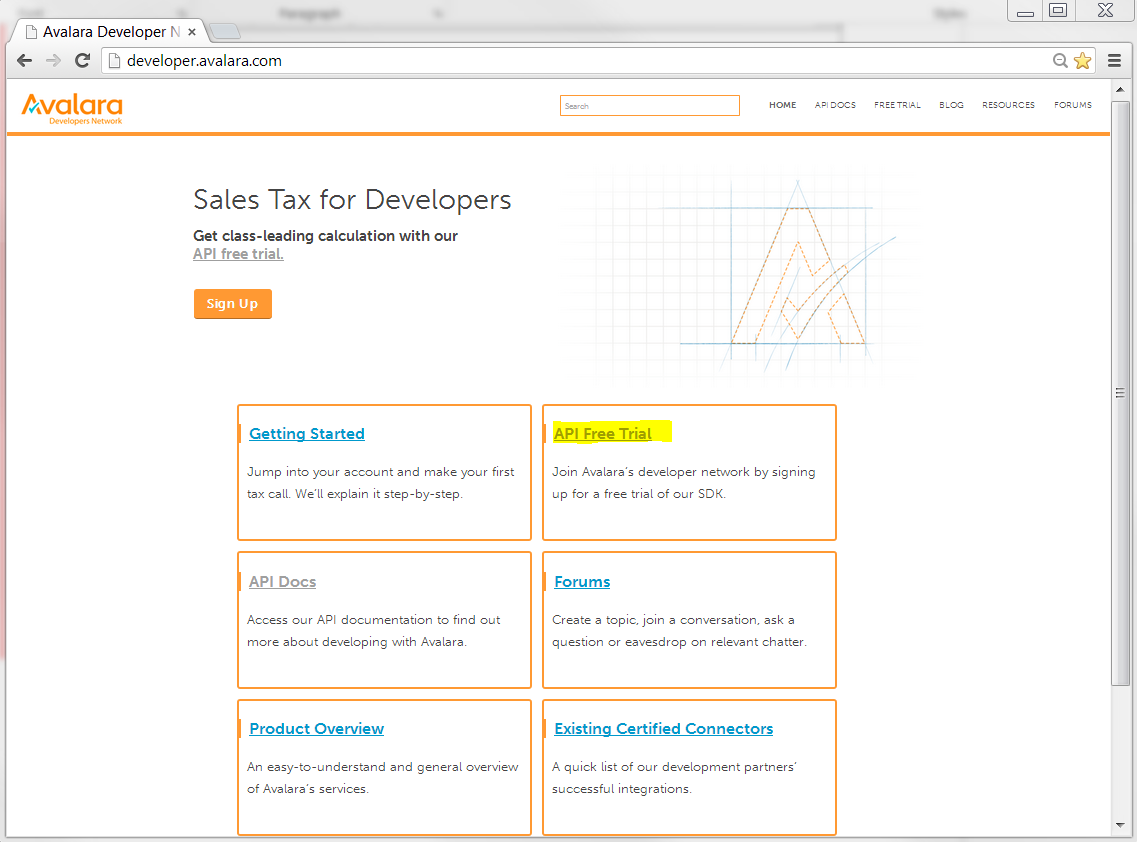
Once you have your Avatax gems installed you can try out the services.

First you will need an Avalara Admin Console account.

<https://admin-development.avalara.net/login.aspx>

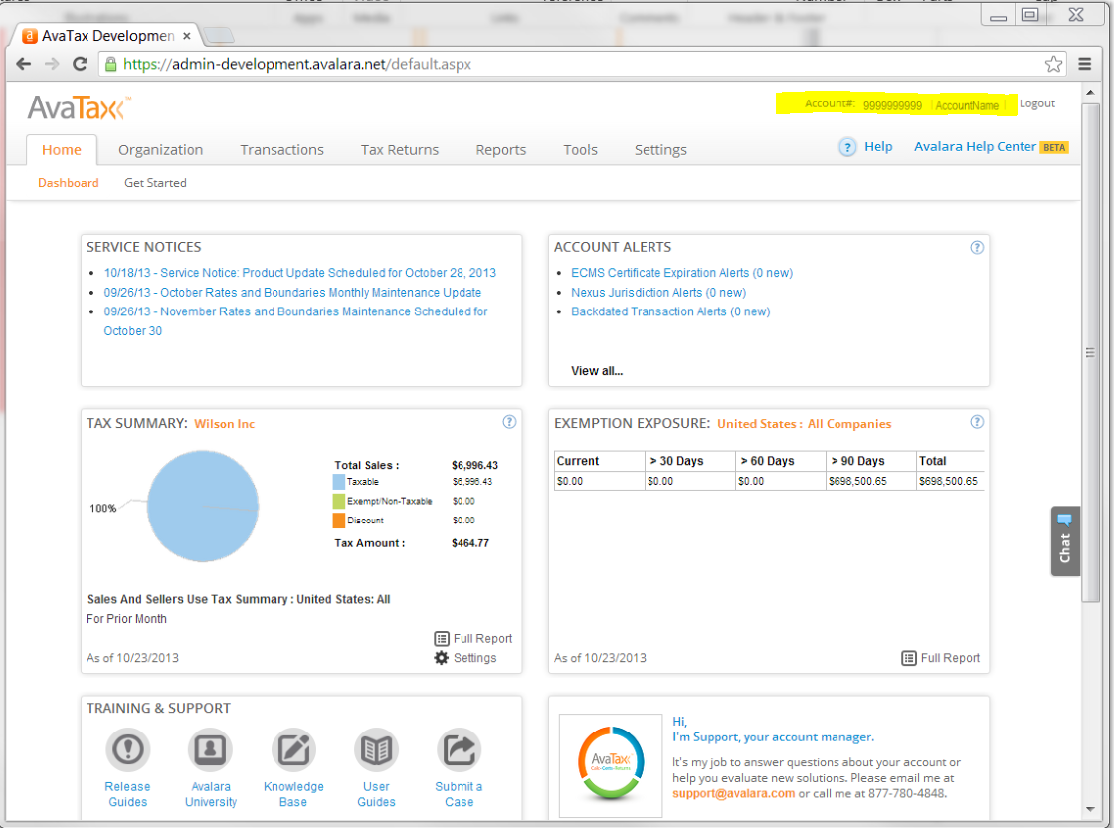


If you don’t already have an account you can get a free trial account from <http://developer.avalara.com/>



Click on the Sign Up button to set up a trial account that will be valid for 1 month. It will be pre-configured for nexus in all jurisdictions and will have a company code of APITrialCompany.

You will receive an account password by e-mail and will be able to sign into your admin console. Note the account number and account name in the top right hand corner. You can use either of these to authenticate to the Avatax web services but we recommend using the account number.



You are now ready to start using the Ruby SDK to consume Avatax web services.

In the test folder of you gem installation you will find some sample programs. We recommend looking at test\_gettax.rb which is a working GetTax call. Just add your username, password and company code and you can run it.

The key elements of using the SDK in your Ruby program are as follows. This example shows a GetTax call.

Require the services

#Load the Avalara Address Service module

require 'avatax\_taxservice'

#Load the Avalara Address Service module - optional

require 'avatax\_addressservice'

Create an instance of the tax service passing your username, password and service details. Also specify whether you wish to access a development or production account.

#Create new credentials hash object

credentials = *Hash*.new

credentials[:username] = 'USERNAME'

credentials[:password] = 'PASSWORD'

credentials[:name] = 'Avalara Inc.'

credentials[:clientname] = ''

credentials[:adapter] = ''

credentials[:machine] = 'Lenovo W520 Windows 7'

* Access Development or Production. Set **:use\_production\_account = true** to access your production account. All other values for this setting will default to Development.

# Determine whether the DEV or PROD service is used. false = DEV true = PROD

credentials[:use\_production\_account] = false

Create a TaxService instance

#Create a tax service instance

TaxServ = *AvaTax*::*TaxService*.new(credentials)

Optionally create an instance of the address service.

#Create an address service instance

AddrService = *AvaTax*::*AddressService*.new(credentials)

Call the GetTax method. The parameters are explained in the section of this manual that details the methods.

#Create new document hash object

document = *Hash*.new

#Populate the fields required by the GetTax call

document[:companycode] = 'APITrialCompany'

document[:doctype] = 'SalesOrder'

document[:doccode] = "MyDocCode100"

document[:docdate] = "2013-10-11"

document[:salespersoncode] = "Bill Sales"

document[:customercode] = "CUS001"

document[:customerusagetype] = ""

document[:discount] = ".0000"

document[:purchaseorderno]= "PO123456"

document[:exemptionno] = ""

document[:origincode] = "123"

document[:destinationcode] = "456"

#Pass addresses as an array of hashes

# <AddressCode>123</AddressCode>

# <Line1>100 Ravine Lane</Line1>

# <Line2/>

# <Line3/>

# <City>Bainbridge Island</City>

# <Region>WA</Region>

# <PostalCode>98110</PostalCode>

# <Country>US</Country>

# <TaxRegionId>0</TaxRegionId>

# <Latitude/>

# <Longitude/>

document[:addresses]= [

{:addresscode => "123",:line1 => "100 ravine lane", :line2 => "Suite 21",:city => "Bainbridge Island",:region => "WA",:postalcode => "98110",:country => "US",:taxregionid => "0",:latitude => "",:longitude => ""},

{:addresscode => "456",:line1 => "9436 NE Blue Wave Ct",:city => "Bainbridge Island",:region => "WA",:postalcode => "98110",:country => "US",:taxregionid => "0"}

]

#Pass order/invoice lines as an array of hashes

# <No>1</No>

# <OriginCode></OriginCode>

# <DestinationCode></DestinationCode>

# <ItemCode>Canoe</ItemCode>

# <TaxCode></TaxCode>

# <Qty>1</Qty>

# <Amount>300</Amount>

# <Discounted>false</Discounted>

# <RevAcct></RevAcct>

# <Ref1>ref1</Ref1>

# <Ref2>ref2</Ref2>

# <ExemptionNo></ExemptionNo>

# <CustomerUsageType></CustomerUsageType>

# <Description>Blue canoe</Description>

# <TaxOverrideType>TaxAmount</TaxOverrideType>

# <TaxAmount>10</TaxAmount>

# <TaxDate>1900-01-01</TaxDate>

# <Reason>Tax Credit</Reason>

# <TaxIncluded>false</TaxIncluded>

# <BusinessIdentificationNo></BusinessIdentificationNo>

document[:lines] = [

{:no => "1",:itemcode => "Canoe",:qty => "1",:amount => "300.43",:discounted => "false",:ref1 => "ref1",:ref2 => "ref2",:description => "Blue canoe",:taxoverridetypeline => "TaxAmount",:taxamountline => "10",:taxdateline => "1900-01-01",:reasonline => "Tax credit",:taxincluded => "false"},

{:no => "2",:itemcode => "Rowing boat",:qty => "1",:amount => "800.12",:discounted => "false",:ref1 => "ref3",:ref2 => "ref4",:description => "Red rowing boat",:taxoverridetypeline => "None",:taxamountline => "0",:taxdateline => "1900-01-01",:taxincluded => "false"}

]

document[:detaillevel] = "Tax" #The level of detail you want returned by the service

document[:referencecode] = "" #Reference code - used for returns

document[:hashcode] = "0" #Set to 0

document[:locationcode] = "" #Store Location, Outlet Id, or Outlet code.

document[:commit] = "false" #Invoice will be committed if this flag has been set to true.

document[:batchcode] = "" #Optional Batch Code

document[:taxoverridetype] = "None" #Type of TaxOverride

document[:taxamount]= ".0000" #The TaxAmount overrides the total tax for the document, if not 0

document[:taxdate] = "1900-01-01" #Tax Date is the date used to calculate tax

document[:reason] = "" #Reason for applying TaxOverride. = ""

document[:currencycode] = "USD" #3 character ISO 4217 currency code (for example, USD)

document[:servicemode] = "Remote" #All lines are calculated by AvaTax remote server

document[:paymentdate] = "2013-09-26" #Indicates the date payment was applied to this invoice

document[:exchangerate] = ".0000" #Indicates the currency exchange rate

document[:exchangerateeffdate] = "1900-01-01" #Indicates the effective date of the exchange rate.

document[:poslanecode] = "" #Optional POS Lane Code

document[:businessidentificationno] = "" #Optional Business Identification Number

document[:debug] = false #Run in debug move - writes data to tax\_log.txt

document[:validate]= false #If true - addresses will be validated before the tax call

Call the Tax Service

#Create empty hash for the tax result details

tax\_result = *Hash*.new

#Call the tax service

tax\_result = *TaxServ*.gettax(document)

The result is returned in the form of a symbol style hash. Here’s an example.

{:get\_tax\_response=>

{:get\_tax\_result=>

{:transaction\_id=>"4119292583172098",

:result\_code=>"Success",

:doc\_id=>"0",

:doc\_type=>"SalesOrder",

:doc\_code=>"MyDocCode100",

:doc\_date=>#<Date: 2013-10-11 ((2456577j,0s,0n),+0s,2299161j)>,

:doc\_status=>"Temporary",

:reconciled=>false,

:timestamp=>"2014-01-21T00:47:38.0520081Z",

:total\_amount=>"1100.55",

:total\_discount=>"0",

:total\_exemption=>"0",

:total\_taxable=>"1100.55",

:total\_tax=>"78.81",

:total\_tax\_calculated=>"94.65",

:hash\_code=>"0",

:tax\_lines=>

{:tax\_line=>

[{:no=>"1",

:tax\_code=>"P0000000",

:taxability=>true,

:boundary\_level=>"Address",

:exemption=>"0",

:discount=>"0",

:taxable=>"300.43",

:rate=>"0.086000",

:tax=>"10",

:tax\_calculated=>"25.84",

:tax\_included=>false,

:tax\_details=>

{:tax\_detail=>

[{:country=>"US",

:region=>"WA",

:juris\_type=>"State",

:juris\_code=>"53",

:tax\_type=>"Sales",

:base=>"300.43",

:taxable=>"300.43",

:rate=>"0.065000",

:tax=>"7.56",

:tax\_calculated=>"19.53",

:non\_taxable=>"0",

:exemption=>"0",

:juris\_name=>"WASHINGTON",

:tax\_name=>"WA STATE TAX",

:tax\_authority\_type=>"45",

:tax\_group=>nil,

:rate\_type=>"G",

:state\_assigned\_no=>nil},

{:country=>"US",

:region=>"WA",

:juris\_type=>"City",

:juris\_code=>"03736",

:tax\_type=>"Sales",

:base=>"300.43",

:taxable=>"300.43",

:rate=>"0.021000",

:tax=>"2.44",

:tax\_calculated=>"6.31",

:non\_taxable=>"0",

:exemption=>"0",

:juris\_name=>"BAINBRIDGE ISLAND",

:tax\_name=>"WA CITY TAX",

:tax\_authority\_type=>"45",

:tax\_group=>nil,

:rate\_type=>"G",

:state\_assigned\_no=>"1804"}]},

:exempt\_cert\_id=>"0",

:tax\_date=>#<Date: 2013-10-11 ((2456577j,0s,0n),+0s,2299161j)>,

:reporting\_date=>#<Date: 2013-10-11 ((2456577j,0s,0n),+0s,2299161j)>,

:accounting\_method=>"Accrual"},

{:no=>"2",

:tax\_code=>"P0000000",

:taxability=>true,

:boundary\_level=>"Address",

:exemption=>"0",

:discount=>"0",

:taxable=>"800.12",

:rate=>"0.086000",

:tax=>"68.81",

:tax\_calculated=>"68.81",

:tax\_included=>false,

:tax\_details=>

{:tax\_detail=>

[{:country=>"US",

:region=>"WA",

:juris\_type=>"State",

:juris\_code=>"53",

:tax\_type=>"Sales",

:base=>"800.12",

:taxable=>"800.12",

:rate=>"0.065000",

:tax=>"52.01",

:tax\_calculated=>"52.01",

:non\_taxable=>"0",

:exemption=>"0",

:juris\_name=>"WASHINGTON",

:tax\_name=>"WA STATE TAX",

:tax\_authority\_type=>"45",

:tax\_group=>nil,

:rate\_type=>"G",

:state\_assigned\_no=>nil},

{:country=>"US",

:region=>"WA",

:juris\_type=>"City",

:juris\_code=>"03736",

:tax\_type=>"Sales",

:base=>"800.12",

:taxable=>"800.12",

:rate=>"0.021000",

:tax=>"16.8",

:tax\_calculated=>"16.8",

:non\_taxable=>"0",

:exemption=>"0",

:juris\_name=>"BAINBRIDGE ISLAND",

:tax\_name=>"WA CITY TAX",

:tax\_authority\_type=>"45",

:tax\_group=>nil,

:rate\_type=>"G",

:state\_assigned\_no=>"1804"}]},

:exempt\_cert\_id=>"0",

:tax\_date=>#<Date: 2013-10-11 ((2456577j,0s,0n),+0s,2299161j)>,

:reporting\_date=>#<Date: 2013-10-11 ((2456577j,0s,0n),+0s,2299161j)>,

:accounting\_method=>"Accrual"}]},

:tax\_addresses=>

{:tax\_address=>

[{:address=>"100 Ravine Ln NE Ste 21",

:address\_code=>"123",

:boundary\_level=>"0",

:city=>"Bainbridge Island",

:country=>"US",

:postal\_code=>"98110-2687",

:region=>"WA",

:tax\_region\_id=>"2109716",

:juris\_code=>"5303503736",

:latitude=>nil,

:longitude=>nil,

:geocode\_type=>"StreetLevel",

:validate\_status=>"NormalHit",

:distance\_to\_boundary=>"0"},

{:address=>"9436 NE Blue Wave Ct",

:address\_code=>"456",

:boundary\_level=>"0",

:city=>"Bainbridge Island",

:country=>"US",

:postal\_code=>"98110-4626",

:region=>"WA",

:tax\_region\_id=>"2109716",

:juris\_code=>"5303503736",

:latitude=>nil,

:longitude=>nil,

:geocode\_type=>"StreetLevel",

:validate\_status=>"NormalHit",

:distance\_to\_boundary=>"0"}]},

:locked=>false,

:adjustment\_reason=>"0",

:adjustment\_description=>nil,

:version=>"1",

:tax\_date=>#<Date: 2013-10-11 ((2456577j,0s,0n),+0s,2299161j)>,

:tax\_summary=>nil,

:volatile\_tax\_rates=>false},

:@xmlns=>"http://avatax.avalara.com/services"},

:"@xmlns:xsi"=>"http://www.w3.org/2001/XMLSchema-instance",

:"@xmlns:xsd"=>"http://www.w3.org/2001/XMLSchema"}

Always check the ResultCode. It will contain ‘Success’ if all went well. If not, you will need to take some kind of corrective action.

#Always check the result code

if tax\_result[:get\_tax\_response][:get\_tax\_result][:result\_code] = "Success" then

puts "The GetTax call was successful"

else

puts "The GetTax call failed"

end

Here’s an example of retrieving the total taxable amount.

TotalTaxable = tax\_result[:get\_tax\_response][:get\_tax\_result][:total\_taxable]

Here’s an example of retrieving the total tax for the invoice.

TotalTax = tax\_result[:get\_tax\_response][:get\_tax\_result][:total\_tax]

## Other Files Included with Your Gem

In your gem ‘lib’ folder as well as the Ruby program you will find the following types of file.

1. .Template Files

The SDK uses the Ruby ERB templating system. Template files have a .erb extension.

<http://ruby-doc.org/stdlib-2.0.0/libdoc/erb/rdoc/ERB.html>

Each web service operation has its own template (e.g. the GetTax service uses template\_gettax.erb).

You should not attempt to modify any of these templates.

1. Log File

This file logs events that occur during the call to the service (e.g. tax\_log.txt). If you call a service with the debug flag set to true the log will contain a much higher level of detail. Debug is great for testing but should be turned off once you go live.

## Process Overview

An example of the simplest integration of an AvaTax connector would be a workstation (or server) that has an application to process retail sales transactions and has access to the internet. All that is needed is an Avalara SDK Connector to process the tax calculations.

To demonstrate a bit more complex example of how an AvaTax SDK connector might be deployed, a “Hybrid” solution is described below.

First of all, there are essentially three basic SDK implementation scenarios referred to in this document:

Standard ERP Connector: An interface that calls APIs from an existing ERP (Enterprise Resource Planning software) or e-Commerce System.

Web Store: The developer utilizes a third party “web store” interface that calls or consumes a custom SDK connector pointed at the AvaTax web service.

Hybrid: The SDK connector is used to retrieve tax results for consumption for an ERP, web store, or retail point of sale system for price quoting purposes. A Standard ERP Connector (described above) for example; MAS 200/500, QuickBooks, Great Plains, AX etc.) later in the business process manages invoicing, saving, posting and committing sales orders or saved documents. See the diagram [Hybrid Model in Appendix C](#_APPENDIX_C:_) for a clearer picture of this process.

## How It All Works

To fully appreciate how an AvaTax SDK connector fits into a typical retail sales process, let’s examine a Hybrid Implementation’s - transaction process from start to finish (once again, See the diagram [Hybrid Model in Appendix C](#_APPENDIX_C_(Hybrid) for a clearer picture of this process).

### Collecting Information

* A potential[[2]](#footnote-2) customer has added items to be purchased from a web store catalog into a “shopping cart” tool[[3]](#footnote-3).
* When the customer is ready to check out, they will want to know the total of their purchase including any shipping charges and the sales tax amounts before making the final decision to commit to a sale (authorizing charges to their credit card; a process outside of AvaTax).
* The customer is either prompted to login using their existing web store credentials, or prompted to complete a customer/user “profile,” resulting in a customer login. Either scenario will produce a high-quality ship to address (or destination address) required by the AvaTax tax engine to ensure a valid tax assessment and result are obtained (see [Address Validation](#_Address_Validation_1)).

### Processing Information

There are two (of four) Document Type (DocType) methods that can be used in a typical GetTaxRequest: SalesOrder and SalesInvoice. Setting the DocType to either SalesOrder or SalesInvoice will result in different outcomes as follows:

* SalesOrder returns results however does not record (or save) the document (or invoice) to your AvaTax data base, and subsequently is not available for posting, committing, tax reporting, or an appearance in your Admin Console’s Transactions listing. The results can be coded to appear similar to the following:

TotalAmount: 953.74

TotalTax: 46.26

Line: 1 Tax: 46.26 TaxCode: P0000000

Juris Type: State;

Juris Name: COLORADO;

Juris Code: 08;

Tax Name: CO STATE TAX;

Rate: 0.029000;

Amt: 27.66

* SalesInvoice does record the transaction to the AvaTax data base, or the Document is in a saved “state” (uncommitted) for further processing (PostTax, CommitTax, CancelTax, etc,) and becomes eligible for tax reporting when committed:

#### We recommend as a best practice to:

* Use DocType = SalesOrder to provide quotes to customers which can be recalled repeatedly without benefit of an invoice number, or an unnecessary document being saved on AvaTax until the customer is satisfied that their order is ready to convert to a sale.
* Use DocType = SalesInvoice when the customer is satisfied that their order is complete, and the credit card process has returned an authorization code. Your connector can proceed with calling “GetTax”—this time with the DocType enumeration set to SalesInvoice.

NoteS

* An uncommitted (saved) document will result and be stored on the account number’s database under the company code[[4]](#footnote-4) called in the GetTax call. The saved transaction will appear in the AvaTax Admin Console, however it will not be available for reporting until the document has been posted and committed (covered later in this document).
* It is possible to call GetTax with the Commit property set to “True” to avoid the subsequent PostTax and CommitTax calls if your business model requires it, or if Post and Commit are unnecessary steps in the process.

### Drop Shipping

Drop Shipping is more of a business process to define vs. a technical solution—one that will require the talents of your company’s tax attorney and/or accountant to define what your nexus liabilities and responsibilities are for the origin and destination locations.

Note

* Avalara cannot provide recommendations or advice regarding how your company should handle nexus liabilities – this is ultimately the responsibility of your tax attorney or accountant.

What needs to be defined (in advance) is if a “nexus event” is taking place or not. The following example illustrates a drop shipping scenario:

* A customer orders a desk and a chair from your web store totaling $1,500.

There is a shipping and handling charge totaling $50 ($25 fee for each, not related to Origin or Destination)

The desk ships from your warehouse in Phoenix (where you have nexus liability)

The chair however is drop shipped from a partner company with a warehouse in New London, CT. (where you do not have nexus).

How does tax get calculated? Here is what we know so far:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Line Number | 1 | 2 | 3 | 4 |
| Item | **Desk** | **Chair** | Shipping **Desk** | Shipping **Chair** |
| Qty | 1 | 1 | 1 | 1 |
| Amount | $ 505.00 | $ 995.00 | $ 25.00 | $ 25.00 |
| TaxCode | P0000000 | P0000000 | FR020100 | FR020100 |
| Origin Address | **800 E Mountain View Ave** | **7462 Kearney St** | **800 E Mountain View Ave** | **7462 Kearney St** |
| Origin Address | **Glendora** | **Commerce City** | **Glendora** | **Commerce City** |
| Origin Address | **CA** | **CO** | **CA** | **CO** |
| Origin Address | **91741-2769** | **80022-1335** | **91741-2769** | **80022-1335** |
| Origin Address | **US** | **US** | **US** | **US** |
| Destination Address | 801 Pontiac St | 801 Pontiac St | 801 Pontiac St | 801 Pontiac St |
| Destination Address | Denver | Denver | Denver | Denver |
| Destination Address | CO | CO | CO | CO |
| Destination Address | 80220-4825 | 80220-4825 | 80220-4825 | 80220-4825 |
| Destination Address | US | US | US | US |

The results of this invoice were as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Item | **Desk** | **Chair** | Shipping **Desk** | Shipping **Chair** |
| Tax Code | P0000000 | P0000000 | FR020100 | FR020100 |
| Sourcing | D | D | D | D |
| Line Amount | $ 505.00 | $ 995.00 | $ 25.00 | $ 25.00 |
| Discount | $ - | $ - | $ - | $ - |
| Non-Taxable | $ - | $ - | $ - | $ - |
| Taxable | $ 505.00 | $ 995.00 | $ 25.00 | $ 25.00 |
| Tax | $ 38.49 | $ 75.81 | $ 1.92 | $ 1.89 |

# Integration Touch Points

## About SDK Sample Code

The sample code provided during pre-sales and with the product downloads are working samples that will provide results as long as they are properly configured and in a runtime environment suitable for the APIs being reviewed. However, they should not be deployed or used as a “connector” or “plug-in” ultimately delivered to the customer as a finished solution. Our samples are populated with numerous pre-populated data elements to make testing the sample in a development environment more meaningful, but they must be modified or removed based on the business needs of the client installation.

## Getting Started

The more complex part of planning your integration with the AvaTax Ruby SDK Adapter is determining:

* **when** to call the AvaTax web service,
* **when** to update your ERP’s internal tables with the results data that is returned from the AvaTax web service,
* **when** to save the results to the AvaTax database in preparation for tax reporting, and
* **how** to handle returns, delayed shipments, credits, and discounts.

These “touch points” where your ERP or e-Commerce system interfaces with the AvaTax Adapter are best organized by the basic operations or APIs[[5]](#footnote-5), some of which are included in the SDK samples.

## Ping

The Ping API provides a means of testing an internet connection and validating AvaTax Credentials (account, license, login, password, etc.). Typical usage is for the connector interface to test connections that appear to be down or unavailable in a new screen or attached to a menu object for the user to call on demand. Typical results would appear as follows:

Ping ResultCode is: Success

Ping Version is: 11.11.1.1

## GetTax

The GetTax API is the core of the Avalara Service. It is capable of collecting a string of elements contained within a single object to include origin (shipped from address), destination (shipped to address), and line items (products, Freight, Service, etc.) information. Individual line item origin / destination addressing is available as well.

Access to the Tax Service API’s is made available by installing the Avatax\_TaxService gem.

GetTax is typically called during a SalesOrder or SalesInvoice DocType transactions.

Note

* It is not a requirement to call a Validate API prior to a GetTax as a base level address validation / “normalization” occurs as described below in AddressValidation -- however it is a best practices recommendation to validate all addresses first to ensure that a quality calculation will result from passing accurate street, city, state, and zip properties through the GetTax method.

The return of a successful GetTax call is at a minimum Total Amount and Total Tax. By setting the DetailLevel property from—for example, “Tax” to “Line”—a much greater set of results will be returned.

A GetTaxResult, when properly coded, would typically appear similar to this:

GetTax is: Success

DocCode: 100811153742

TotalAmount: 1000

TotalTax: 86

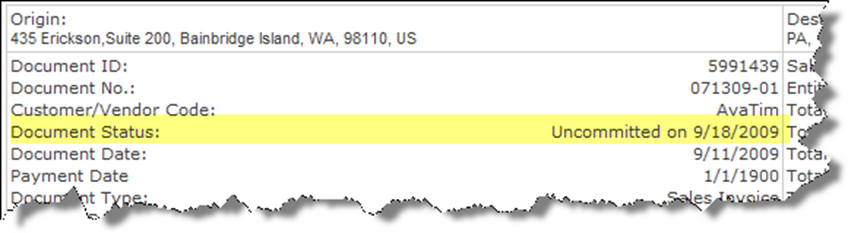
Line: 1 Tax: 86 TaxCode: P0000000

Juris Type: State; Juris Name: WASHINGTON; Rate: 0.065000; Amt: 65

Juris Type: City; Juris Name: BAINBRIDGE ISLAND; Rate: 0.021000; Amt: 21

## PostTax and CommitTax

The PostTax and CommitTax APIs can be used to set the AvaTax Document[[6]](#footnote-6) status to several states during a SalesInvoice process (remember, SalesOrder DocTypes are not saved). AvaTax documents appear initially as “uncommitted” on the Admin Console and do not appear at all on Avalara reporting until they are committed.



Once an invoice is posted via a PostTax call, its Document Status is changed to Posted on mm/dd/yyyy.

A posted document can be replaced with any subsequent GetTax calls to the same DocCode (Document Number), which has the effect of changing the Document Status back to Uncommitted.

CommitTax completes a two stage process and sets the Document Status to Committed on mm/dd/yyyy. However, it cannot be overwritten as the PostTax operation was (with exception for some AvaTax ERP Connectors).

The typical integration point for the PostTax/CommitTax process is when it is a requirement of the particular ERP or a business process to flag document events, for example when multiple levels of approval are required prior to a document being set to a committed / reportable event.

Committing an invoice should be performed at whatever point an invoice will no longer need to be changed or modified in any way by the ERP. Any changes that must be done after committing must be completed via an adjustment function such as a CancelTax (voids the document) or a Return Invoice.

Note

* Some ERPs (e.g. Dynamics GP) have separate posting and committing steps. Others (e.g. QuickBooks) consider an invoice committed when the invoice is saved. However, in this case, a committed document can be overwritten based on a set of business rules specific to the QuickBooks Connector.

As stated above, Posting and committing a document is a two-stage process designed to ensure both the client’s ERP and AvaTax documents are committed via a process that mirrors most ERP systems. It is not, however, a requirement to follow this process, if it does not meet your business needs (for example, see [GetTax Post / Commit = True](#_Address_Validation)).

## NewDocCode Property

AvaTax Ruby SDK products may use the NewDocCode property.

|  |  |
| --- | --- |
| Property | Notes |
| DocCode | * The customer’s internal reference code used by the client application, generally a sales or invoice number. * DocCode is a unique identifier {string value} that all documents saved on the AvaTax database which will coincide with the reference code used by the client application. * When this field is not populated, a GUID or Globally unique identifier represented as a 32-character hexadecimal string is passed as the DocCode.   For example: 21EC2020-3AEA-1069-A2DD-08002B30309D |
| NewDocCode | As of version 5.7 and greater, you can change an existing document’s DocCode to a new DocCode during a PostTax or CommitTax function. This property eliminates the problem of changing an existing sales order number or GUID to a client controlled tracking number for customers requiring this business functionality. |
| DocID | A unique Document number used only by Avalara. DocID was removed from TaxSvc methods and properties as of version 5.7 but still appears on the Admin Console for backwards compatibility reasons. DocID should not be considered in new SDK integrations past version 5.7. |

Example

Your web store saved a document in the AvaTax database using a GUID 0b26f690-0e8d-4e84-b3a3-b5ef33d3134f . You would like to change this to an internal Invoice number 100001.

Solution: During a PostTax operation on the GUID in question, you add:

newdoccode = ‘100001’

…which would then become your new DocCode number.

## Committing a Document—a “Two Stage Process”

Taking into consideration the elements described above, the steps for committing a saved or uncommitted document via the two-stage process are described below. This process assumes that you know the following data elements:

* Invoice Number (Document Code)
* NewDocCode
* Total Invoice Amount
* Total Invoice Tax
* Doc Date (yyyy-mm-dd)

### Stage One

* The client’s ERP should start a PostTax routine (may be called something different depending on the ERP deployed) which will show the invoice posted locally (to the ERP). Simultaneously a PostTax call to the AvaTax web service should be made using elements collected above.

### A success message should result and should be passed to the ERP to indicate that both the ERP and AvaTax databases are in sync. A failure or “NOT” success (Boolean) should generate aStage Two

* The client’s ERP should start a CommitTax routine (may be called something different depending on the ERP deployed), which will show the invoice as committed to the local ERP’s general ledger. Simultaneously a CommitTax to the AvaTax web service should be made using elements collected above.
* A success message should result and should be passed to the ERP to indicate that both the ERP and AvaTax databases are in sync. A failure or “NOT” success (Boolean) should generate a reconciliation process.

### Failure Situations

* If the client side PostTax fails, then the document cannot be Committed on the AvaTax database. Correct the problem and attempt a PostTax again. If a CommitTax is called on an un-posted document, a DocStatusError will result.
* If the client-side CommitTax fails, then the document either failed to post, or something is wrong with the CommitTax variables (the wrong DocCode, or the Company Code should be reviewed.)
* If a document was improperly committed, the client side should call a [CancelTax](#_CancelTax) to “un-post” the document (Voided State), thereby removing it from all Sales Tax Reporting.
* Error messages should be logged with the document number so follow-up can be completed. See [Appendix B](#_APPENDIX_B:_) for common system messages.

## CancelTax

CancelTax provides a mechanism to recover from posting problems, or completely removes a document from tax reporting. The effect of a CancelTax depends upon the current state of the document (uncommitted, saved, posted, or committed).

* A document with a state of saved is one which has not been posted or committed (last operation = GetTax). Calling CancelTax on such an invoice has no effect.
* A document with a state of posted is one which has been posted but not committed (last operation = PostTax). Calling CancelTax on such an invoice reverts it to a state of saved. This means the document can have its state changed to posted again at a later time. Posted documents can be overwritten by subsequent GetTax calls.
* An invoice with a state of committed is one which has been committed (and appears in Sales Tax Reporting). Calling CancelTax on a committed invoice has the effect of removing it from reporting, but moves it to a fourth state: Voided.
* An invoice in a state of uncommitted is one which will be found in the Admin Console, indicating a document/invoice that has not been posted or committed.

A voided invoice cannot have its status changed (and the invoice number cannot be re-used).

## GetTaxHistory

GetTaxHistory provides a means to report or retrieve data on historical documents.

For SDK developers, GetTaxHistory provides a mechanism to write custom reports. One such use, valuable to some ERP systems, is to create a custom tool to post and commit invoices that have ended up in a state of posted in the ERP program, but not posted (showing in reports) in the AvaTax Admin Console.

GetTaxHistory returns both the original GetTaxRequest object, as well as the GetTaxResult object. The GetTaxResult object shows the current state of the invoice in the AvaTax system.

Sample result:

:ResultCode=>["Success", "Success"] #The original request and result status

:GetTaxResultDocId=>["36260589"] #The invoice or order number

:GetTaxResultDocStatus=>["Saved"] #The current document status

## Validate

The Validate API uses the *Avalara.AvaTax.Adapter.*AddressService namespace (as opposed to the *TaxSvc* namespace) and processes an object containing an address, returning either success with minor unambiguous address corrections, or a failure code message indicating whether or not the address submitted is a valid US or Canadian street address.

The Address service can be installed using the Avatax\_AddressService gem.

See **test\_address.rb** for an example of how to use the address validation service.

The Validate API is typically used to validate an address book (customers) and in some cases to produce properly formatted shipping labels.

Calling a Validate with a minimum of three elements of an address is required. For example, line 1, 2, and/or 3 of the street address, the city, and the state, will produce several outcomes:

* If the validation is successful, a “normalized” address is returned. For example;

900 winslow way, bainbridge island, 98110

…will return

900 Winslow Way E, Bainbridge Island, WA, 98110-2450

* If the validation is unsuccessful, an error message will be returned with some indication of why it failed. For example:

Validate ResultCode is: Error

AddressRangeError: The address number is out of range

* Typical scenarios that result in errors are:
  + Missing address information (e.g. zip code),
  + Abbreviations that do not conform to USPS standards (e.g. Bvrd vs. BLVD, or wash vs. WA, suite vs. ste, etc.).
  + Street Address out of range (e.g. 100 main street vs. 110 Main Street).

# Required Elements, Limitations, and Security

## Client Profile

This is a required element in your AvaTax SDK integrations for many very good reasons.

* The values passed in aid in troubleshooting connector problems.
* It identifies OEM transactions for developer/partner programs.
* It provides the developer with a method to version their adapters and locate them in an enterprise environment.

You should pass appropriate values for the following.

name = Your company name (e.g. Avalara Inc.)

clientname = Your application name (e.g. WebMarket)

adapter = The adapter release you are using (e.g. Ruby SDK 1.0.1)

machine = The platform you are running on (e.g. Window 7 Server)

## 

## Number of Lines per Document

At this writing, there is a limit of **1000** lines per document. Improving our scalability, including handling of large documents, is a key deliverable for future releases, and we are making significant changes to increase the performance and reduce the contention between our web service callers. We will notify developers via version release notes as they become available.

Currently the best way to avoid documents that have more than 1000 lines is to make use of the batch services function for large documents. If you have additional questions about this issue, contact [*sdksupport@avalara.com*](mailto:sdksupport@avalara.com).

## 

## Web Services Security

### AvaTax Web Service and Proxy Server Settings

Our web service uses the standard SSL (secure socket layer) on port 443 with the addresses <https://development.avalara.net> for development and <https://avatax.avalara.net> for production. As SOAP traffic during API calls—In and Out—is handled during the API call; the AvaTax web service *does not* contact the client server at any time. Therefore, all you need to do is add one or both of the URLs to your proxy server configuration to allow outgoing calls.

### Connecting to the AvaTax Service and DNS Lookups

The way that client applications connect to Avalara’s services is through our published URL - <https://avatax.avalara.net/>. This URL is translated to an IP address by DNS (Domain Name System) behind the scenes. It is critical that all AvaTax clients use this published URL instead of the corresponding IP address when connecting to the AvaTax service. The reason for this is that Avalara may change the IP address associated with the URL at any time, without notice, as part of our practice of load balancing our services across multiple data centers and/or across multiple ISPs.

Therefore, to maintain *uninterrupted* access to the AvaTax web service, the client must implement best practices around DNS access. Otherwise, transactions destined for an outdated IP address (from a localhosts file for example) will fail and will not be protected by Avalara’s SLA (service level agreement).

Client adapters must respect the Time To Live (TTL) setting associated with the DNS record (60 seconds). Hence, any adapter, environment variables or configurations that “cache” the IP address *longer* than the TTL interval is considered to not be following best practices for accessing Internet-based SaaS products.

#### Clients Using a “Static” Host File Table:

Avalara’s web services do not support host files. The system accessing our service must use the DNS to look up IP addressing. Failure to do so will put the clients adaptor at risk in a failover situation (meaning our primary web service has been switched to a secondary due to an outage or preventative maintenance cycle).

#### Clients Deploying Security Enforced Firewall Rules:

If your company deploys a security practice that requires locking down outbound /inbound traffic to a range (or set) of IP addresses, those addresses are available by request to [support@avalara.com](mailto:support@avalara.com). However, these addresses do change over time, so it is recommended that you use the DNS names of development.avalara.net or avatax.avalara.net to resolve this issue.

# Best Practices & Tips

## GetTax / PostTax Commit Method

The **GetTax** and ***PostTax*** methods may commit documents directly using the **Commit** property via Boolean values (the default value is *false*) when set determines the *state* of documents being saved.

If the property is set to *true*, the document’s state is set to committed, thus eliminating the need to make separate web service calls to the PostTax and CommitTax methods for tax reporting.

NOTE:

* Use with care! Once a document is committed, it cannot be modified other than via the CancelTax method (see [Document States](#_AvaTax_Document_States)).

## Posting and Committing Invoices (Documents)

The recommended procedures to post an invoice are as follows:

* **Call PostTax** for the **DocCode** (or your invoice number). This action sets the Document *status* to *posted* and records the date it occurred. The document appears as *uncommitted* in the Admin Console.

   
  
NOTE:

* PostTax takes the invoice **Total Amount** and **Total Tax** as arguments . These values are used to verify that the invoice amounts in the ERP agree with the invoice amounts in the AvaTax database. If they disagree, a warning message displays, but this will not prevent the completion of the posting process. Therefore, in the event of a ResultCode = SeverityLevel.Warning, it is advisable that the DocCode (your invoice number) be reported to a log, and then reconciled with the ERP via another process.

If the PostTax issuccessful, proceed with posting theinvoice within your ERP system.

Call *CommitTax* for the invoice when appropriate. CommitTax completes the process for AvaTax reporting. The document will then appear as *committed* in the Admin Console.

NOTE:

* Both GetTax and PostTax can be called with the CommitTax property set to “True” to bypass the two stage process of post and commit.

## AvaTax Document States

|  |  |
| --- | --- |
| **Document State** | **Occurs when…** |
| Saved | * The last call was to **GetTax**, or… * The last call was to **CancelTax** with the **PostFailed** flag set, or… * The document was in a state of **Posted** when a **CancelTax** was called. |
| Posted | * The last call was to **PostTax** |
| Committed | * The last call was to **CommitTax** (**Note**: only invoices in a **Committed** state appear on reports.) |
| Voided | * The last call was to **CancelTax** with the **DocDeleted** flag set |

* For a **GetTax** to succeed, the document must either not exist (i.e., a new invoice), or be in a state of saved. If the document is in any other state, the Resultcode will be equal to Error, and the message returned will be “DocStatus is invalid for this operation.” Successful calls leave the document in a saved state.
* For a **PostTax** to succeed, the document must be in a state of saved. A successful call leaves the document in a posted state.
* For a **CommitTax** to succeed, the document must be in a state of posted. A successful call leaves the document in a committed state.
* For a **CancelTax** with a **PostFailed** flag to succeed, the document must be in a state of posted. A successful call leaves the document in a saved state.
* For a **CancelTax** with a **DocDeleted** flag to succeed, the document can be in any state, except voided (saved, posted, committed). A successful call leaves the invoice in a voided state.

NOTE:

* There is no way to move an invoice from a state of voided to any other state.

## Detail Level Enumeration

In a **GetTaxRequest** (or **GetTaxHistoryRequest**) the *DetailLevel* specifies the level of tax detail to return to the client application following a tax calculation.

|  |  |
| --- | --- |
| **Member Name** | **Description** |
| Summary | Reserved for future use. |
| Document | Document-level (GetTaxResult) details. TaxLines will not be returned. |
| Line | Line-level details (includes Document details). TaxLines will be returned but TaxDetails will not be returned. |
| Tax | Tax jurisdiction-level details (includes Document, TaxLines, and TaxDetails). |
| Diagnostic | In addition to Tax-level details, indicates that the server should return information about how the tax was calculated. Intended for use only while the SDK is in a development environment. |

How results would appear depends upon the *DetailLevel* set during the GetTax call.

Example Results:

**Document:**

GetTax Result: Success

DocCode: 34567.2

TotalAmount: 1010

TotalTax: 84.34

**Line:**

GetTax Result: Success

DocCode: 34567.2

TotalAmount: 1010

TotalTax: 84.34

     Line: 1 Amount: 1000 Tax: 83.5

     ItemCode: AvaItem-001 TaxCode: P0000000

**Tax:**

GetTax Result: Success

DocCode: 34567.2

TotalAmount: 1010

TotalTax: 84.34

     Line: 1 Amount: 1000 Tax: 83.5

     ItemCode: AvaItem-001 TaxCode: P0000000

          Juris Type: State; Juris Name: COLORADO; Rate: 0.029000; Amt: 29

          Juris Type: County; Juris Name: ADAMS; Rate: 0.007500; Amt: 7.5

          Juris Type: City; Juris Name: COMMERCE CITY; Rate: 0.035000; Amt: 35

          Juris Type: Special; Juris Name: SCIENTIFIC & CULTURAL FAC.(CD); Rate: 0.001000; Amt: 1

          Juris Type: Special; Juris Name: METRO FOOTBALL STADIUM DISTRICT (FD); Rate: 0.001000; Amt: 1

          Juris Type: Special; Juris Name: RTD GREATER DENVER; Rate: 0.010000; Amt: 10

Where the *DetailLevel* setting becomes especially important is in the tax details that are returned from countries that make use of GST, PST, and HST – for example Canada.

* **GST** = General Sales Tax (appears in AvaTax **CNT** or **Country**)
* **PST** = Provincial Sales Tax (appears in AvaTax **State**)
* **HST** = Harmonized Sales Tax (**State -** combined similar to SST)

These values would be returned in the Tax *DetailLevel* in the following manner:

GetTax Result: Success

Address: **1919 Upper Water St, Halifax, NS B3J 3J5**

DocCode: 34567.2

TotalAmount: 1010

TotalTax: 84.34

     Line: 1 Amount: 1000 Tax: 151.00

     ItemCode: AvaItem-001 TaxCode: P0000000

          Juris Type: State; Juris Name: NOVA SCOTIA; Rate: 0.10; Amt: 100 (or PST / HST)

          Juris Type: County; Juris Name: CANADA; Rate: 0.050; Amt: 51 (or GST)

## Handling Return Invoices

Return Invoice processing is largely a business practice that the developer and the business manager need to map out prior to coding or moving forward to a production environment. As the business needs of every company is as unique as their products, it should not be expected that the scenarios described below will work for your specific situation, or that we have any one size fits all solution that you can snap in to your development. Rather, it is up to the developer to fashion a solution that fits the company’s business model.

A thorough understanding of the SDK Adapter’s GetTaxRequest members and [document states](#_AvaTax_Document_States) is key to developing an architecture suitable for processing returns in your development environment. Different business needs require different solutions.

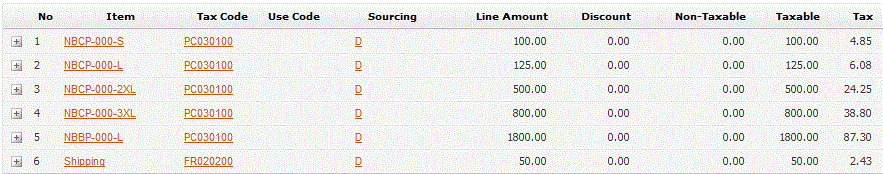
Example:

For the example below, we will process a return invoice with the following assumptions:

* The document has already been committed and tax remitted to the tax jurisdictions.
* There may be multiple lines in the document.
* A complete or partial refund is the expected outcome.

We will process a return on the following invoice:

#### OriginalInvoiceAdminConsole.GIF

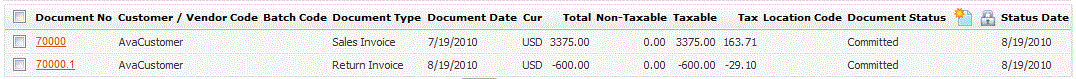
With the following lines items::  
****

Steps:

* Call a **GetTax** with a duplicate of the document (invoice) you want to process returns on:
  1. Using the same invoice number, or
  2. Using a new invoice number (DocCode) with the original invoice number passed in the reference number field, or
  3. Re-using the original document’s invoice number with a “.1” added.

NOTE:

* You can only use the ReturnInvoice method once when using the original DocCode. A DocStatusError will result if you attempt to commit a second document using ReturnInvoice with a duplicate DocCode.



Set the **DocType** to ReturnInvoice (very important if you are re-using the original DocCode)

Set the **DocDate** to the tax reporting month that you want the return to appear in (typically the current month).

Set the **TaxOverride** property to TaxDate

Set the **TaxOverrideDate** to the date the original invoice computed tax. AvaTax will calculate the tax based on this date.

Set the **AdjustmentReason** to Return Items, or whatever makes sense for your business needs.

**Optional:** Set **GetTaxRequest.ReferenceCode** = to the original Invoice for tracking purposes.

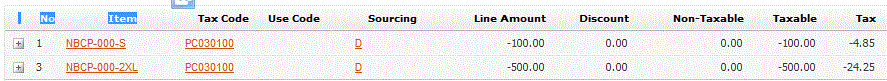
NOTE:

* Pass only **line items** being returned -- do not include the line items that will not be returned unless all items are returned). In the example below we are returning line items 1 and 3.

Set the **Amt** property to a negative dollar amount

Note:

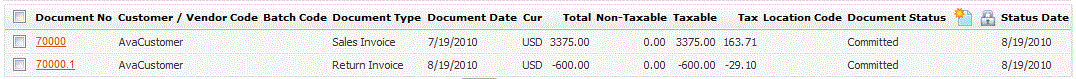
* Always leave **QTY** as a positive/neutral number.



Once sent to the AvaTax web service, the tax engine will return negative tax amounts on the line items based on the **TaxDate** specified. If no TaxDate is set, the document date will be used to calculate tax.

NOTE:

* The Taxable Amounts on a Return Invoice show negative amounts equal to the items returned.

  
Summary:

Out of the six items originally processed, two items have been reversed on your tax reporting. In this case, $29.10 appears essentially as a “credit” on the current month’s tax liability.

## Handling Shipping and Freight

Shipping (Freight) should be sent as a separate line item on Documents (invoice). Use the pre-defined tax code of **FR020100** (**GetTaxRequest.TaxLines[X].TaxCode**). This tax code will automatically charge or not charge tax on Freight, according to the regulations of the state referenced in the Ship To address.

Handling Discounts (or Gift Certificates)

The recommended best practice for handling discounts or Gift Certificates is as follows:

* It is expected that the GetTax call will pass a dollar amount of the discount at the Document level.
* It is also expected that one to many line items will have the Discounted property set to “True”.
* For each line item to which the discount applies, AvaTax will allocate the discount across those line items and reduce the tax base by the amount of the allocation.
* On the Admin Console, the line items will show the Discounted amounts applied (see image below).
* For line items that you want discounted, ensure you pass the *net amount* (vs. the discounted amount) to the Adapter.

NOTE:

* Manufacturing Discounts should not be passed to AvaTax.

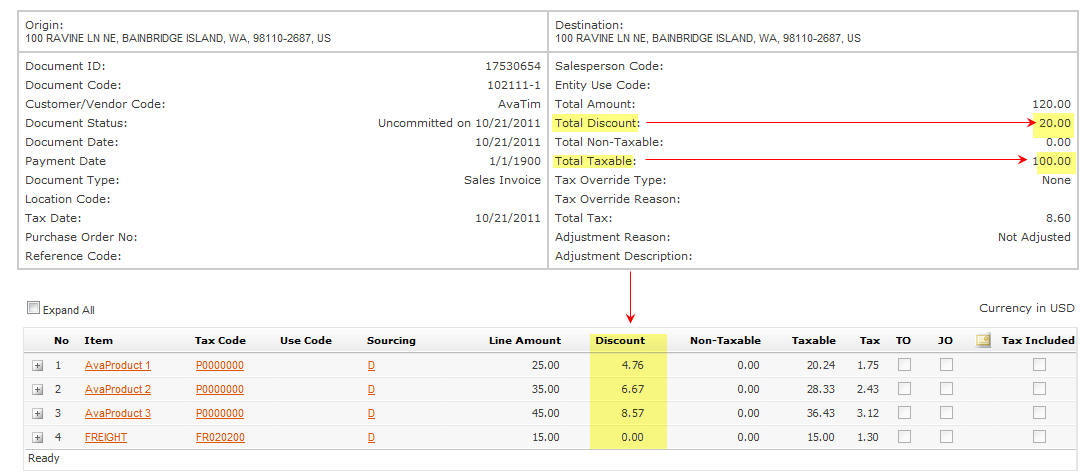
Examples:

* **Basic**: A $25.00 line item is sold with a $10.00 discount. The resulting tax ***base*** will be $15.00.

**Simple**: Two $25.00 line items are sold with a $10.00 discount, with only *one* item marked for the discount. The resulting tax base will $15.00 for the item marked for Discount and $25.00 for the item *not* marked for discount.

**Typical:** Two $25.00 line items are sold with a $10.00 discount, with both line items marked for the discount. The resulting tax base will $20.00 for each item marked for discount.

**Complex:** Three line items and a Freight line item are sold (one for $25, $35, and $45) with a **$20.00** discount, with all items (*except Freigh*t) marked for the discount. The resulting tax calculation would appear this way on the Admin Console.



NOTE:

* The discount was applied in equal portions across the three items but not on the Freight.
* The taxable amount was reduced by a total of the discount applied ($20)

## Error Messages

Understanding and taking appropriate action on error messages is critical to the successful implementation of a custom SDK connector. The errors and warnings that result from web service calls will guide the receiver to the most logical problem resolution path.

For example; if an API, such as GetTax fails, the receiver should always iterate through the messages collection that will be part of the result set and check the contents of the *Name*, *Summary*, *Details* and *RefersTo* fields of each message. Often the contents of these messages will indicate what the problem is.

For example, below is a SOAP trace from a DocumentNotFoundError:

<Messages>

<Message Name="DocumentNotFoundError">

<Summary>The tax document could not be found.</Summary>

<Details>SO123456789</Details>

<RefersTo />

<Severity>Error</Severity>

<Source>Avalara.AvaTax.Services.Tax</Source>

</Message>

</Messages>

All API calls should contain a path that will execute for:

* ResultCode = SeverityLevel.*Success*
* ResultCode = SeverityLevel.*Warning*
* ResultCode = SeverityLevel.*Error*
* ResultCode = SeverityLevel.*Exception*

NOTES:

* Typically, **Exception** is only returned by a system error from the AvaTax web service and will be caused by circumstances outside of the client application’s control (i.e. not having an internet connection will return the error).
* In general, the path taken for ResultCode= **Exception** should be the same as for ResultCode=**Error**.
* The only known circumstance under which ResultCode=**Warning** would be returned is for the API PostTax when the TotalAmt or TotalTax fields do not agree with the amounts in the Avalara Database.
* At this time, the only API that may return SeverityLevel.**Warning** is PostTax. To account for the possibility that this may change in the future, we recommend you have a code path that will execute for ResultCode SeverityLevel.Warning for all API calls. This codepath may initially be the same as the codepath that is executed for a ResultCode of SeverityLevel.Error.

The following error message:

* “The server was unable to process the request due to an internal error. For more information about the error, either turn on IncludeExceptionDetailInFaults (either from ServiceBehaviorAttribute or from the <serviceDebug> configuration behavior) on the server in order to send the exception information back to the client, or turn on tracing as per the Microsoft .NET Framework 3.0 SDK documentation and inspect the server trace logs.”

*…*is returned frequently when one of the following has occurred:

* There is a malformed data element (SOAP) being passed in the GetTax call, for example the date 10/01/2010 vs. 2010-10-01 or DocType= *salesinvoice* vs. *SalesInvoice.*
* There is a network/internal (local service) interruption preventing the web service call to contact the AvaTax web service.
* There is a configuration problem with your adapter host settings.

## Address Validation

Address validation is a difficult process to pin down with regards to the results that may be returned, as we are guided by the addresses that appear in the United States Postal Service database. For example, if a new address (new construction within the past six months) has not been updated in this database, the AvaTax engine will not be able to return a normalized address as below, because it simply does not exist.

900 winslow way, bainbridge island, 98110

…will return

900 **W**inslow **W**ay **E**, **B**ainbridge **I**sland, **WA**, 98110**-2450**

This section is primarily dedicated to gaining a practical understanding of the validation process and possibilities to help you develop a business practice that will return the most accurate results.

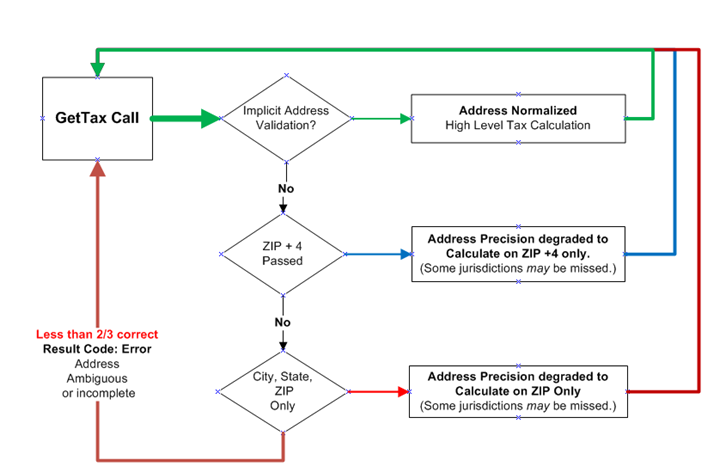
It is important to understand that there are two different types of address validations that can be performed:

* ***explicit*** - which are performed as a *stand-alone* address validation calls (*Validate* method)
* ***implicit*** - which occur during a tax calculation (*GetTax* method).

**Explicit** address validations are *standalone* address validation calls (meaning address validation only – no tax calculations) that are passed to the AvaTax web service, then during the same call (or port opening), results are returned with either a *success*—if no ambiguous address information was passed*—*a *normalized* address. Otherwise an error message is returned (i.e., *AddressRangeError*, as in the example below) indicating the problem and summary details related to the address submitted.

A Typical Validate Address Error Results message would look like this:

**Name**: AddressRangeError  
**Severity**: Error  
**Summary**: The address number is out of range  
**Details**: The address was found but the street number in the input address was not between the low and high range of the post office database.  
**Source**: Avalara.AvaTax.Services.Address  
**RefersTo**: Address.Line1

**Implicit** address validations, on the other hand, are a little more subtle and are tightly coupled to the AvaTax *GetTax* method’s logic. For example, when you make a GetTax call to the AvaTax web service, the tax calculation engine attempts to resolve the address with the best possible address data that can be gleaned from the address data that was provided. The logic follows the flow chart below: 

Logically, to trap whether an address passed is good enough for the document in question, one only needs to collect the results on the zip code. Was it **Zip Code Plus** “55428-1234” or just a **5 digit Zip Code** “55428”—this should help you decide if a follow up is necessary.

Best practices:

It is a best practice to call [***Validate***](#_Validate) on addresses when they are entered initially or edited. However, it is not a requirement. The [GetTax](#_GetTax) address validation deploys its own trimmed down address validation process for origin and destination addresses prior to determining taxing jurisdictions.

Which method meets your needs?

The ***GetTax* Address Validation** method seeks to *normalize* (or make correct) these addresses prior to using them in tax calculations. It does not, however, return the normalized address as part of the *GetTaxResults* class. If that is the desired result; use the *Validate* method. See the [*Address Validation - GetTax Method*](#_Address_Validation_-_2)for more detailed information.

A successful***Validate*** call returns the normalized address as part of the results class and is considered shipping label ready, given US Postal Service standards are maintained. The behavior of the **Validate** method is to return *ValidateResult.ResultCode* of *success* or *not success*.

* If ResultCode is **Success**, then the address submitted contains the normalized and completed version of the input address, and the *ValidateResults.Messages* property is empty (no errors to report).
* If ResultCode is **Error**, then the address submitted is an empty collection, and *Messages* contains the information indicating why the address did not pass validation.

For example;

**Validate ResultCode is: Error**

**AddressUnknownStreetError: An exact street name match could not be found**

See Appendix B for a list of common error messages.

What if the address will not validate (success)?

Even if an address will not validate, it may still be usable in a GetTax calculation. The GetTax method does not require a full street address to be able to accurately determine taxing jurisdictions. While a fully validated street address is always better and more accurate, taxing jurisdictions can usually be unambiguously determined if, at a minimum, *city*, *state* and *zip* *code* are provided (and at least 2/3 are valid).

* The [GetTaxResult](#BoundaryLevels) value returned will indicate what level of the address GetTax used to determine the applicable taxing jurisdictions (*Address*, *Zip9* or *Zip5* , or *BoundaryLevel* 0,1 and 2 respectively). *Jurisdiction boundary precision level* depends on the accuracy of the address as well as the precision level of the state-provided jurisdiction boundaries.  
    
  If an incomplete address is given, and GetTax succeeds, the address contained enough information for *GetTax* to *unambiguously* determine applicable jurisdictions. Note that sometimes, even if a complete and validated address is sent to GetTax, the BoundaryLevel field may be populated with a lesser value than Address. This means that zip5 or zip9 was enough precision for GetTax to be certain of the jurisdiction.
* The efficiency of address validation being entered by customers, or even an invoicing clerk, can be maximized by performing the validation via input fields.
  + **For example,** a zip code should be of the pattern ‘#####’ or ‘#####-####’ for US postal codes, and ‘C#C #C#’ for Canadian Postal codes, where;

# = a digit (0-9), and

C = an alphabetic char (A-Z).

* A way of ensuring a proper city abbreviation is used is to employ the use of a *pick list* or a *drop-down list* for States and Provinces.

### **Address Validation - GetTax Method**

While a fully validated street address is always better, taxing jurisdictions can usually be unambiguously determined if, at a minimum, city, state and zip code are provided (and at least 2/3 are valid).

However, even though it is possible for the GetTax Request to be called devoid of a valid street address or even zip code, the GetTaxResults may often omit Special taxes (i.e., Transit, stadium, levy, etc.) that are specific to the jurisdictional boundaries that the shipped to address may be liable for. Without a valid street address, it may be next to impossible to return tax results that should be included if the address was less ambiguous.

A GetTax API will attempt to normalize an address, but it does not return the normalizedaddress as part of the GetTaxResults class. If the address cannot be normalized, it will continue with processing a Sales Order/Invoice with what it has. Albeit rare, incomplete or erroneous addresses submitted via GetTax operations will return incorrect results.

For example, on a sale of a $125 item with $25 Shipping to Commerce City, Colorado, the results based on the Zip Code would be as follows:

|  |  |
| --- | --- |
| Commerce City, CO, 8002**1**  returns a total tax of $14.81  (80021 is actually Federal Way) | Commerce City, CO, 8002**2** (which is correct)  returns a total tax of $ 8.51 |

**Note** that both GetTax calls were successful based on State and ZIP Code.

AvaTax’s ability to successfully identify boundaries is multi-tiered:

* “Base tax” based on the street address – the highest level or Boundary Level 1.
* If not street address, the base tax is based on the “ZIP9” or ZIP Code +4 (Boundary level 2 returned).
* If not ZIP9, then the standard ZIP Code is used. (Boundary level 2 also returned).
* The GetTaxResult.BoundaryLevel value returned (0, 1, 2)  will indicate what level of the address the GetTaxResult used to determine the applicable taxing jurisdictions - Zip5, Zip9, Address or 2,1,0 respectively.

Note that sometimes, even if a complete and validated address is sent to GetTax, the BoundaryLevel field may be populated with a lesser value than Address. This means that zip5 or zip9 was enough for GetTax to be certain of the local jurisdiction. Anything less will result in one of the following error messages and will stop the GetTax call from succeeding:

* **RegionCodeError** or the two-digit state code was not specified or was invalid. CountryError or the country name or code was not recognized.
* **TaxAddressError** or a taxable address must include a (line, city, and region (state)) or a (line and zip) or a (region (state) and postal code).
* **AddressRangeError** or the address on line one was found but the street number in the input address was not between the low and high range of the post office database.
* **PostalCodeError** or the ZIP/Postal Code does not exist and could not be determined by the city/municipality and state/province.

If an incomplete address is given and GetTax succeeds, the address contained enough information for GetTax to *unambiguously* [[7]](#footnote-7)determine applicable jurisdictions.

See Appendix B for some of the common error messages.

# Tax Compliance

AvaTax tax calculations are *data-driven*, meaning data that affects a particular tax calculation may be:

* State and Local Nexus Admin Console settings,
* Taxability rules based upon Customer type codes and Item tax codes
* Exemption Certificates
* Other tax rules defined.

NOTE:

* SST and AvaFile customers are required to send a value in the GetTaxRequest.Lines[n].**ItemCode** and **Description** fields.

## What is NEXUS?

Nexus is the *sufficient* connection a business has with a taxing jurisdiction. This connection obligates the business to calculate, collect, report, and remit tax. The connection is established by virtue of the business activity conducted in the taxing jurisdiction.

NOTE:

* Contact your accountant, tax attorney, or other qualified sales tax professional to conduct a nexus study for your business. Avalara does not provide advice or consulting on nexus studies.

Typically, a business must have a *substantial physical presence* in a taxing jurisdiction to have nexus there. The following items may constitute a substantial physical presence in a taxing jurisdiction, and therefore an obligation to calculate, collect, report, and remit tax:

* A corporate office, storefront or remote sales office
* Remote employees working from home on company payroll
* Owned or rented warehouse space containing owned inventory
* Owned inventory leased to a customer
* Sales or marketing representatives making regularly scheduled visits
* Delivery of product by a company owned vehicle

Rules governing what constitutes nexus vary from jurisdiction to jurisdiction. Determining exactly how a rule applies to your business is critical.

**AvaTax assumes:**

* Transactions in nexus jurisdictions are fully taxable
* Transactions in non-nexus jurisdictions are non-taxable

More granular control over tax results is provided via the Admin Console by managing a combination of the following:

* Nexus
* Tax Codes
* Tax Rules
* Items to Tax Code Mapping
* Exemption Certificates

If you have more questions, please refer to your Admin Console Setup or contact Avalara for Admin Console Training.

## ItemCode vs. TaxCode and Taxability Rules

Special product taxability rules can be defined using ItemCode and/or TaxCodes, i.e., Line.ItemCode or Line.TaxCode.

* *ItemCodes* represent individual products, whereas *TaxCodes* represent categories of products.

For example…  
  
ItemCodes of **M** for Milk, **E** for Eggs, and **C** for Cheese.  
  
…then a TaxCode of **GR** might be established for the broader category of Groceries.

*Product Taxability Rules* are defined in the Avalara Admin Console in terms of TaxCodes. For example, groceries are not taxable in the State of Washington, so a taxability rule might be established that states “when a line item is passed with the Tax Code GR, and the destination address is Washington, the line item is charged $0 tax.”

The mechanism to get the tax rule defined above is by one of two methods:

1. In the Avalara Admin Console, map ItemCodes to TaxCode(s), andthen send the ItemCode for your individual Product SKU.  
     
   Using the example above, ItemCodes **M**, E**,** and **C** would be mapped to each via TaxCode **GR**. Then if Line*.*ItemCode = **M**, **E** or **C**, they would automatically map to TaxCode **GR**, and the rule for TaxCode GR will fire.
2. Send the TaxCode GR in the Line.TaxCode field.

Method #1 is popular, because it allows you to send your own internal SKUs without setting up any additional tables. This is especially true for PRO or SST customers, because Avalara is typically supplying the TaxCode(s) for them.

If you send a value in BOTH the ItemCode and the TaxCode fields, the behavior is as follows:

* If the ItemCode sent maps to a TaxCode, the **mapped** TaxCode will be used for product taxability rules, regardless of what is sent in the TaxCode field.
* If the ItemCode sent does not map to a TaxCode, the TaxCode sent will be used for product taxability rules.
* Lines that have—or are mapped to—TaxCodes that have no rule associated with them are simply taxed at the applicable rates for Origin, Destination, and Nexus settings.

# Going Live (Production)

As part of our **SDK Support Services** we provide a “code review,” which amounts to reviewing the data and the “tax profile” on your Admin Console from your development account to ensure requirements for the service(s) you have selected are being met, and that you are receiving the results that you expect from our service. Performing this type of development and testing in a production environment can be time consuming and costly.

See [**Appendix E: SDK Analysis Check List**](#_APPENDIX_E:_SDK) for details

## Production Credentials

Once the review of test data is completed, and any issues discovered are addressed, we will issue you Production Credentials. We would like to open the door to having a pre-productionconference call with your accounting and development staff, to ensure all questions and concerns are answered.

## Migration Services

If you would like us to migrate your development Admin Consolesettingsto your new production account, let us know. We would be happy to do this as part of Avalara’s service to you.

## System Outages

Let’s face it… even the best equipment and systems are fallible. The internet is subject to a variety of disabling events that can result in reduced bandwidth or even service outages. Avalara has invested heavily in developing strategies to ensure that our AvaTax web services stay online 24 x 7 x 365—ready to process tax service calls despite conditions of the internet.

Although AvaTax outages have been historically infrequent and short in duration, it is something that should be considered when designing your AvaTax SDK integration. We strongly encourage you to develop your own contingency plans to minimize the effect of these unexpected outages on the flow of your e-commerce system.

### Frequently Asked Questions:

**Q: Does Avalara queue our transactions when the web service is down?**

**A:** No. Unless a connection is made to the web service, you will only receive a HTTP 500 - Internal Server Error message which means:

* You have lost connection to the AvaTax Web Service
* Your local system needs to track invoices until the web service is restored.
* Avalara sends out regular service alerts if it is a problem on our end.

Consideration must then be given to how you will report taxes back to the web store customers during the outage, and the solution sets used are wide and diverse, depending on the ERP / Shopping Cart system used. All of these solutions will require your company’s tax accountant or attorney agreement.

Examples include:

* Charging a “typical” base rate to all transactions appropriate to the product, and then refund the difference to the customer once the transactions can be posted to the AvaTax service.
* Charging no tax and assuming the liability on to your own company.
* Put the shopping cart contents into a queue until connection is restored. Then process the invoice – emailing the customer with the results.
* Shutting your WebStore down for maintenance temporarily.

**Q: What if my connection is down for hours or even days? How can I process my transactions for the date the taxes were collected?**

**A:** The AvaTax Admin Console utility can be used to import transactions that have been processed (by your own ERP) to the committed state. We have easy-to-use comma-delimited or Excel spreadsheet templates that can be populated and imported. This topic is covered in detail in your Admin Console setup manual, or by clicking the **?** in the Admin Console **Import** function.

# Resources

## Avalara University Free Training

Taxation as well as other Company Organizational settings is managed within the AvaTax Admin Console (a secure (SSL) website). Tax Reporting is also a function of the AvaTax Admin Console.

It is a recommended best practice that individuals who will be using the AvaTax Admin Console Utility—as well as the Developers of SDK custom connectors—attend free Avalara University AvaTax webinars available to customers, by going to <http://www.avalara.com/avalarauniversity> .

You can sign up for classes on the Avalara University Product Training Page. Please select the class that is appropriate for your company’s subscription (Pro or Basic).

Developers, although not necessarily maintaining company settings within the Admin Console account, are highly encouraged to take these same webinars to gain a thorough understanding of the Admin Console, and how it relates to the results and actions of the SDK custom connectors.

## Admin Console, Web Service, Customer Center Access

### Admin Console Login

* Enter the URL to the site:
  + Development Admin Console <https://admin-development.avalara.net/>
  + Production Admin Console the URL is <https://admin-avatax.avalara.net/>
* Enter the username and password that was sent to you via email
  + This password is temporary until reset.
  + If you have lost this password, follow the directions provided at the Forgotten password link on the login screen.

### Web Service Access (for the connector API)

Note:

This is not a user access portal. Credentials are sent to the Account Admin only.

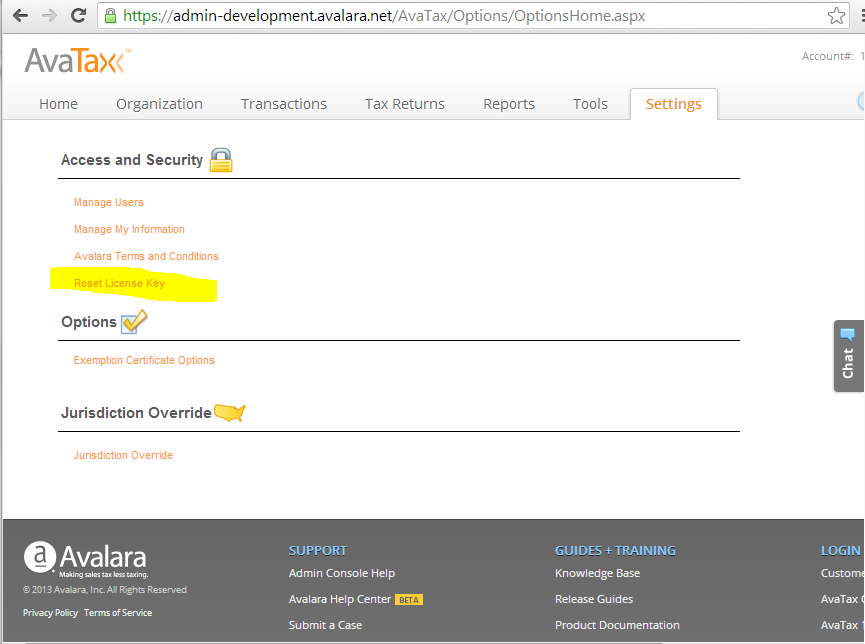
* <https://development.avalara.net> is the URL of the development account service. This must be called out in your SDK connector API.
* <https://avatax.avalara.net> is the URL of the production account service. This must be called out in your SDK connector API.
* Account Number : 1100012345 [Sample] This is the account number that needs to set in the credentials portion of your connector. Tthis is not an Admin Console login.
* License Key: 1A2BC3D4E5F6G7 // [Sample] This is the license key that needs to be set in the credentials portion of your connector. This is not a Admin Console Password.
* AvaTax WSDL is viewable at <https://development.avalara.net/Tax/TaxSvc.wsdl>

## Product Download

<http://developer.avalara.com/api-docs/api-sample-code>

## Resetting your License Key

For security reasons, Avalara Support does not reset a license key for any account. However, your administrator may do so from the Settings tab of the Admin Console. Click the Reset License Key link.



When a license key is “reset”, the new key is emailed to the Account Admin(s) on this list and no one else. This policy prevents service interruption by individuals who might not know the impact of their action. The effect of resetting a license key is the sole responsibility of the Company (Client) Account Administrator.

Warning

* Resetting the license key has the effect of invalidating the old key, and breaks all future connections to the AvaTax web service still using the old license key (including AvaTax ERP Connectors using the same account number.) In other words, when you reset your license key, you will need to change the credential properties on any and all connector(s) you have built, installed and/or deployed, but also the AvaTax ERP connector using the same account number.

## SDK Product Documentation

Avalara ships reference documents with all of our SDK Products. The location of these reference materials depends on where the AvaTax product was installed. Below is a guide for locating reference materials:



# APPENDIX A: (Required Fields)

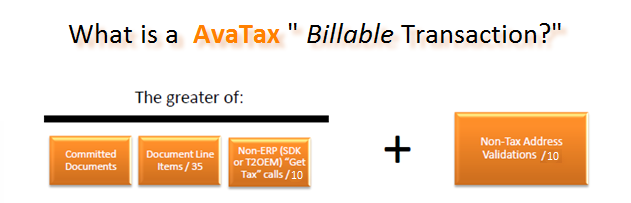
|  |  |  |  |
| --- | --- | --- | --- |
| GetTax Members | Type/Size | Required | Description |
| **DocCode** | String (50) | Required | Invoice, return, or credit memo number, must be unique at the company level:  \**Blank field will result in a* ***DocStatusError*** *Message (see appendix B)* |
| **DocDate** | Date (10) | Required | Invoice, return, credit memo date, Format: \*Blank field will result in a **InternalServiceFault** (see appendix B under "Date") |
| **CustomerCode** | Text (50) | Required | Code utilized within your organization’s ERP/ eCommerce application to identify a customer. Also used in the Certification process. \*Blank field will result in a **RequiredError** (see appendix B) |
| **LineNo** | Text (10) | Required | Transaction line number, must be unique sequence for each document, suggested – sequential number:  1, 2, 3, 4, etc. \*Blank field will result in a **RequiredError** (see appendix B) |
| **ItemCode** | Text (50) | Required | Item Code/SKU identifying the product or service being sold to your organization’s end-customer. \*Blank field will result in a **Warning: Missing argument** (see appendix B) |
| **Amount** | Number | Required | Total sale amount for the line item (Quantity \* Unit Price) \*Blank field will result in a **Warning: Missing argument** (see appendix B) |
| **DestAddress** | Text (50) | Optional With Caveats | Destination/Ship-to street address. To receive a top quality tax result, the street address must be validate-able. \*Blank field will result in a **RequiredError** (see appendix B) |
| **DestCity** | Text (50) | Optional With Caveats | Destination/Ship-to city: To receive a top quality tax result, the City must be validate-able with the Zip Code |
| **DestRegion** | Text (2) | Required | Destination/Ship-to state/province 2 character abbreviation \*Blank field will result in a **TaxAddressError** (see appendix B) |
| **DestPostalCode** | Text (10) | Required | Destination/Ship-to postal code, five character (12345) or nine character (12345-6789) \*Blank field will result in a **TaxAddressError** (see appendix B) |
| **OrigAddress** | Text (50) | Optional With Caveats | Origin/Ship-from street address. To receive a top quality tax result, the street address must be validate-able. |
| **OrigCity** | Text (50) | Optional With Caveats | Origin/Ship-from city. To receive a top quality tax result, the City must be validate-able with the Zip Code |
| **OrigRegion** | Text (2) | Required | Origin/Ship-from state/province 2 character abbreviation \*Blank field will result in a **TaxAddressError** (see appendix B) |
| **OrigPostalCode** | Text (10) | Required | Origin/Ship-from postal code, five character (12345) or nine character (12345-6789) \*Blank field will result in a **TaxAddressError** (see appendix B) |

# APPENDIX B: (Common System Messages)

| Message | Summary | Documentation |
| --- | --- | --- |
| **CompanyNotFoundError** | Company not found. Verify the CompanyCode. | An unknown CompanyCode was specified. All companies must be configured using the Admin console. |
| **DocStatusError** | DocStatus is invalid for this operation. | GetTax expects the document to not exist or the DocStatus to be Saved. PostTax expects the DocStatus to be Saved. CommitTax expects the DocStatus to be Posted. |
| **DocTypeError** | DocType is invalid | An unsupported document type was specified. It must be SalesOrder, SalesInvoice, PurchaseOrder, PurchaseInvoice, ReturnOrder, ReturnInvoice |
| **DocumentNotFoundError** | The tax document could not be found. | The tax document specified could not be found. |
| **RegionCodeError** | Invalid or missing state/province code. | The two-digit state code was not specified or was invalid. |
| **CountryError** | Unknown country name or code | The country name or code was not recognized. |
| **TaxAddressError** | Address is incomplete or invalid. | An incomplete or invalid origin or destination address was given. RefersTo indicates which. A taxable address must include a (line, city, and region (state)) or a (line and zip) or a (region (state) and postal code). |
| **OutOfBalanceWarning** | Document was posted, but is out of balance. | A document was posted in which the specified TotalAmount or TaxAmount did not match the saved document. This is just a warning. The document was posted. |
| **JurisdictionNotFoundError** | Unable to determine the taxing jurisdictions. | Unable to determine the taxing jurisdictions. |
| **AddressRangeError** | The address number is out of range. | The address was found but the street number in the input address was not between the low and high range of the post office database. |
| **InactiveCompanyError** | Tax operations not allowed for an inactive company | Company Code is set to inactive. |
| **DuplicateLineNoError** | Duplicate line number | One or more line numbers are duplicates. All line numbers must be unique within the document. |
| **TaxRegionError** | The TaxRegionId was not found |  |
| **TaxOverrideError** | Tax override cannot be applied. |  |
| **AddressError** | Unable to validate the address. |  |
| **InsufficientAddressError** | Insufficient address information | You must specify at least Line/ZIP, or Line/City/State |
| **PostalCodeError** | Invalid ZIP/Postal Code. | The ZIP/Postal Code does not exist and could not be determined by the city/municipality and state/province. |
| **UnsupportedCountryError** | Country not supported. | Address Validation for this country not supported. |
| **Messages.DateRangeError** | Start Date cannot be later than the End Date | An invalid date range was specified. RefersTo indicates the property. |
| **Messages.RequiredError** | Required element is missing | A required property was not provided. RefersTo indicates which property. This value must be specified. |
| **Messages.UniqueConstraintError** | This is a duplicate of an existing unique element and cannot be saved. | Please verify the element is unique. |
| **Messages.MaxCountExceededError** | MaxCount limit exceeded. Please use filters to limit the result set. | 1000 invoice lines is the current limit. |

# APPENDIX C: (Hybrid Model)HybridModel120109.gif

# APPENDIX D: Billable Transactions SDK



“Transaction” means an electronic request submitted by your connector/adaptor to the AvaTax Web Service to calculate tax (DocType *SalesOrder*), save a document (DocType *SalesInvoice*), or Validate an address. AvaTax calculates your transaction usage on a daily basis (00:00:00 to 23:59:59 GMT) using the following formula:

* **Document Count**; the *greater* of the following:
  + the count of *unique* documents that you commit or void within the Service
  + the count of line items contained in those documents divided by 35
  + the count of tax calculation requests submitted by any SDK connector(s) attached to your account divided by 10

**Plus**

* **Standalone Address Validation**s count divided by 10. This does not include the address normalization process conducted by the GetTax methods.

# APPENDIX E: SDK Analysis Check List

1. **Documents**

* Evidence of committed documents in the Admin Console within the past 60 days preceding the analysis. This applies to “tax results” customers as well for Avalara to provide quality analysis.
* Evidence that a Freight/Shipping line was tested within a saved document.
* Evidence of a voided document (use of CancelTax)
* Evidence that returns, partial returns, delayed shipping, and if appropriate – drop shipping methods were tested successfully.
* Evidence that Return Invoices are passing negative *amounts* vs. negative *quantities*.

1. **Item Codes and Description**

* Verification that there are Item Codes with corresponding Item Descriptions. (a *requirement* for SST and **AvaFile** customers).
* Verification that tax rules, tax codes and / or Item Codes mapped to tax codes are tested in transactions and are functioning as expected – those that appear of character will be flagged and the developer notified via the analysis process.
* **Pro Tax service** subscribersshouldverify that pro tax codes are used during the testing process.

1. **Tax Codes**

* Verify that all Items Codes are not set to the default tax codes. For example; item codes mapped to *Tangible Personal Property or P0000000.*
* Verify that the use of the FR tax code (Freight) is not defaulting to the Tangible Personal Property (P0000000) tax code which *may* have the effect of incorrect tax results.

1. **NEXUS**

* Verify the company’s Local nexus selected (the tax jurisdiction they are doing business in)?
* Verify that the nexus settings appear to support the expressed or written business needs of the customer?
* Evidence that the customer has tested transactions that fall inside and outside of nexus jurisdictions.

1. **Server Audit**
   * Evidence of the use of DocType=“SalesOrder” is present and results in a saved document (DocType=”SalesInvoice.”
   * An analysis that results in an “*appropriate*” ratio of Validate and *SalesOrder* to *SalesInvoice* transactions. Best practice recommendation is ratio to be 2-6-1 (i.e. 2 validates **and** 6 SalesOrder transactions to complete 1 committed document [reportable]).
   * An analysis that results in an “appropriate” number of errors (i.e. address validation errors, or tax service errors encountered in a normal business flow).
   * Evidence that the developer has not simply used downloaded “sample code” as a solution (i.e. profile.client not modified, DocCode using “Sample” and date, Item and descriptions that are appropriate to the customers business needs, etc.)
   * Evidence that the Profile.client has been modified to reflect: identifier information via the TaxSvc.Profile.Client property, where Profile.Client is a string of the form "MyERP,Majver,MinVer,MyConnectorName,Majver,MinVer”

* Verification that future production ready company codes are not flagged as Test companies. Test companies will be reported as part of the analysis results.

# Avalara Support

## How to Get Help with SDK Integration

Avalara’s SDK Support Team is available for consultation via the following methods:

* Phone: 877-780-4848, option 2
* Email: [SDKsupport@avalara.com](mailto:SDKsupport@avalara.com)
* Chat: Log into your Avalara Customer Center and choose “Chat with Support”
* Web to Case: Log into your Avalara Customer Center and choose “Submit a Case”

# Index

*.NET*, 26, 30

Address Validation, 26, 29, 42

AddressRangeError, 29, 41

AdjustmentReason, 23

Admin Console, 32, 33, 35, 36

Amt, 23

AvaFile, 32

Boundary Level, 29

CancelTax, 20

code review, 35

codepath, 25

Commit Method, 19

Committed, 20

CountryError, 29, 41

Credentials, 35

Destination, 34, 40

development, 22, 30, 35

Direct SOAP Calls, 30

DocCode, 19, 22, 40

DocType, 22, 26, 41

document level, 24

document *states*, 22

Documentation, 41

DocumentNotFoundError, 25, 41

Error, 20, 25, 35

ErrorMessages.xls, 27, 28, 30

Exception, 25

Exemption Certificates, 32

FR020100, 23

*Framework 3.0*, 26

Freight, 23

fully taxable, 32

GetTaxResult, 28, 29

Going Live, 35

HTTP 500, 35

Hybrid, 43

*IncludeExceptionDetailInFaults*, 26

ItemCode, 32, 33, 34, 40

Java, 4

jurisdictional boundaries, 29

line items, 23

lines items, 22

Lines per Document, 16

Local Nexus, 32

Messages, 25, 28, 41, 42

Migration, 35

negative dollar amount, 23

NEXUS, 32

non-taxable, 32

normalized, 28, 29

Origin, 34, 40

Outages, 35

PHP, 4, 30

port 443, 16

*PostalCodeError*, 30, 42

posted, 20, 36, 41

PostTax, 19, 20, 25, 41

production, 17, 21, 35

proxy, 17

QTY, 23

Ref1, 23

Ref2, 23

*RegionCodeError*, 29, 41

*ResultCode*, 19, 25, 28

Return Invoice, 21

Return Invoices, 21

SalesInvoice, 26, 41

*SalesOrder*, 41

saved, 20, 41, 42

secure socket layer, 16

*ServiceBehaviorAttribute*, 26

Shipping, 23, 29

SOAP, 17, 25, 30, 31, *See* WSDL

SSL, 16

substantial physical presence, 32

Success, 25, 28

Support, 35

Tax, 19, 25, 32, 33, 41, 42

taxability rules, 32, 33, 34

*TaxAddressError*, 29, 40, 41

TaxDate, 23

taxing jurisdictions, 28, 29, 41

TaxOverride, 23

TaxOverrideDate, 23

*The server was unable*, 26

Training, 33

Transaction, 40

two-digit state code, 29, 41

URL, 17

Validate, 27, 28

Voided, 20

*Warning*, 19, 25, 40

wrapper, 30

WSDL, 30

Zip5, 28, 29, 30

Zip9, 28, 29, 30

1. [↑](#footnote-ref-1)
2. “Potential” because no money has changed hands. [↑](#footnote-ref-2)
3. A shopping cart tool allows online shopping buyers to place items in the cart. Upon checkout, the software typically calculates a total for the order, including shipping and handling (i.e. postage and packing) charges and the associated taxes, as applicable [↑](#footnote-ref-3)
4. Short name for a company name – (i.e. General Electric E Division could be GEEDiv) [↑](#footnote-ref-4)
5. Application Program Interface is an interface implemented by a software program which enables it to interact with other software [↑](#footnote-ref-5)
6. AvaTax Documents are the Sales Invoices that you are processing via the GetTax API for eventual tax reporting. [↑](#footnote-ref-6)
7. Meaning there are no other close matches that may fit more than one address or jurisdiction [↑](#footnote-ref-7)