TLC 2.0 RESTful Interface Notes

Intended Audience

This document is intended for the software developer tasked with interfacing the TLC bookkeeping engine with other applications by using TLC's RESTful interface.

General Approach

TLC offers a RESTful interface for posting most document types to the books of account. The data for these documents will have come from your other systems such as a Sales Order Processing system, an electronic interface to your suppliers and so forth.

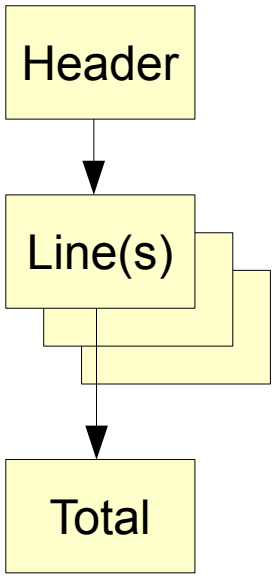
In general, you create a map containing the data of the document you wish TLC to post to its ledgers and then, using JSON, pass this data to the TLC server as an HTTP request using the POST method. If the data is acceptable you will receive a 200 (success) status code response back from the server where the body of the response contains the document number as created by TLC. If an error occurs, the response from the server will have a failure status code (e.g. 400) and the body of the response will contain an error message.

If you look in the web-app/documentation directory of the tlc Grails project, you will see a program called RestTest.groovy that can be run from the operating system command line and which illustrates the above technique. This sample program will be described in more detail later in these notes but, suffice it to say here, that the data in RestTest.groovy is 'hard coded' in since it is only intended as a proof of concept. The main issue is the structure of the data that you pass to the server, and this is described in the following section.

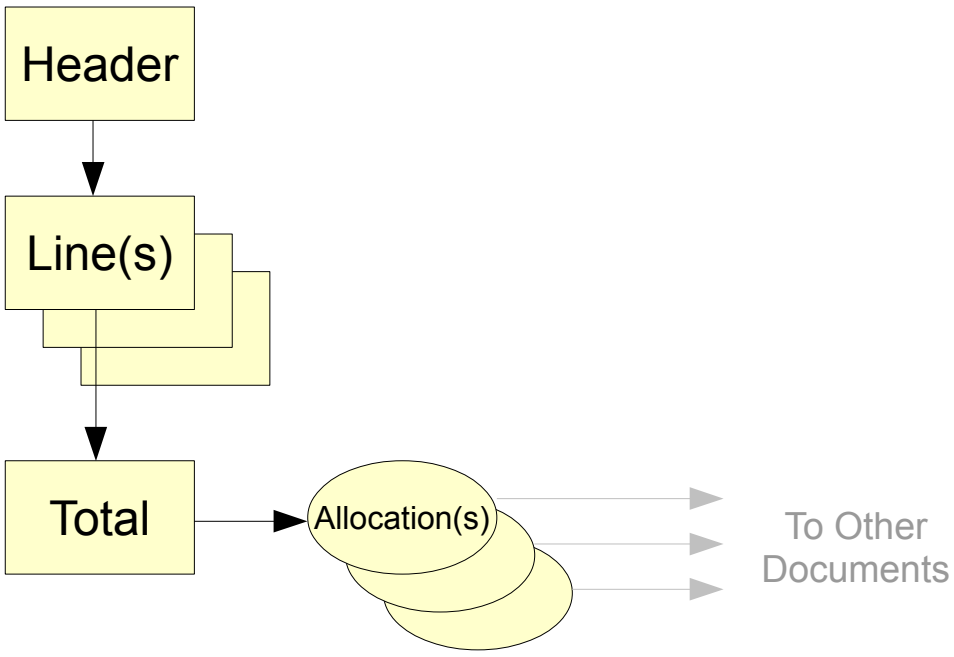
Data Structure Overview

The general outline for creating data that represents a document to be posted by the bookkeeping engine is a single Groovy map that contains one header element, a Groovy list of one or more line elements and an optional total element, where each element is itself a map.

Every document has a single header. Each document has one or more lines. Documents other than self-balancing types (i.e. Journals) have a single total line. Journal type documents do not have a total line. Accruals and Prepayment document types are considered to have a total line. This situation is illustrated by the following diagram:



Any line (including the total line, if there is one) that is to be posted to an Accounts Receivable or Accounts Payable account may have one or more allocations associated with it. An allocation specifies a document and value that the line is to be set off against within that subsidiary ledger account.



The above example shows a total line, say for a Sales Credit Note, being allocated to other documents (presumably one or more sales invoices) on the same customer account.

Based on the above overview, a symbolic representation of the map that you would create within your application as the data to be sent to the server is as follows:

```
def documentData = [header: [map of header fields],
                    lines:  [list of maps of line data],
                    total:  [optional map of total data]]
```

The above 'documentData' map would then be converted to JSON and submitted to the server as a request to create and post a document using the given content. Note that the 'lines' list and the 'total' map can, if that line/total is being posted to an AR or AP account, contain a list of allocations, thus:

```
total: [field1: data1, field2: data2, allocations: [list of allocation maps]]
```

The above example shows how a Groovy list of 'allocation' maps can be included in a total line (or ordinary line if applicable) to specify the documents and amounts that this document line should be allocated against.

The next section gives details of the fields that are allowed in the header, line, total and allocation maps.

Header Map

There can only be one header element in the main map. The fields are as follows:

Key	Data Type	Comments
type	String	This is the code of a document type within the company. You might have document type codes such as 'SI' for sales invoice, 'ESI' for export sales invoice etc. Note that you can not specify a document type that is an Accrual Reversal or Prepayment Reversal since these are automatically generated document types. Nor can you specify a Foreign Currency Difference or Foreign Currency Revaluation document type, for similar reasons.
date	String of Date	Format is 'yyyy-MM-dd'. The date of this document. This is used to determine which accounting period the document will be posted to.
currency	String	Upper case three character code of the currency that this document is in. If this value is not specified, it will default to: a) the currency of the customer/supplier account for invoices and credit notes; b) the currency of the bank/cash account if a payment or receipt; c) the company currency if neither of the foregoing is applicable.
due	String of Date	Optional. Format is 'yyyy-MM-dd'. Only applies to sales and purchase invoices and, even then, only if you want to override

Key	Data Type	Comments
		the standard terms for the customer/supplier.
reference	String	Optional reference 'number' of up to 30 characters. Typically used for the supplier's invoice number etc.
description	String	Optional description of up to 50 characters explaining what this document is about. The description you enter here, if any, is automatically copied to the total line, if there is one and if it does not have a description of its own.
hold	Boolean	Optional and only applicable to lines (including total lines) being posted to an AR or AP account. Set to true if you want to place the line on settlement hold (i.e. not to be paid or allocated against).
adjustment	Boolean	Optional and only applicable to General Ledger Journal type documents and provisions. Set to true if you would like this document treating as an adjustment value rather than as a normal trading value.
turnover	Boolean	Automatically set to true for invoice/credit note document types. For other document types, this is optional and need only be set to true if the line/total is an AR/AP account line and you would like the value to be treated as affecting the customer/supplier turnover figures for the year.

Line Map(s)

The main map must contain a key of 'lines' which has a value that is a list. The list must contain at least one entry. Each entry in the list represents a line in the body of the document. Note that, with the exception of Journal document types, monetary values within a line map should be in the 'normal' form. For example, a GL analysis line with a value of 100.00 from within the body of a purchase invoice would be taken to be 100.00 debit analysis to (say) a GL expense account whereas, if the document had been a sales invoice instead, it would be taken to be a 100.00 credit analysis to (presumably) a GL revenue account. Journals by contrast use positive values to mean debit and negative values to mean credit.

Key	Data Type	Comments
ledger	String	One of 'gl', 'ar', 'ap' to specify the ledger that this line is to be posted to.
account	String	The full (case sensitive) account code, within the ledger specified above, to which this line is to be posted.
description	String	Optional description of up to 50 characters explaining what this line is about.
value	BigDecimal	The monetary value of this line excluding any tax that may be associated with the line.

Key	Data Type	Comments
tax	BigDecimal	Optional. The monetary value of any tax associated with this line. Only sales/purchase invoices/credit notes and petty cash payments/receipts can have line level tax values.
code	String	If a tax value is specified above, then the tax code under which the line is being taxed (e.g. 'exempt', 'standard' etc.)
auto	Boolean	Optional and only applies to a line being posted to an AR or AP account. If set to true, an auto allocation will be performed on the account after any specific allocations (see below) have been performed.
allocations	List	Optional and only applies to a line being posted to an AR or AP account. A list of allocation maps to set this line off against other documents on the same supplier/customer account.

Total Map

For all document types except Journal types, a total line is used to confirm that the sum of the lines is correct when compared to the total line. The total line also supplies additional information such the account to which the total is to be posted. Like the line maps described above, the value of a total line is specified in its 'normal' form. Thus a purchase invoice with a total value of 100.00 would represent a credit to the supplier account whereas, if the document were a sales invoice instead, it would represent a 100.00 debit to the customer account. The fields are as follows:

Key	Data Type	Comments
ledger	String	One of 'gl', 'ar', 'ap' to specify the ledger that this total line is to be posted to. For an accrual or prepayment type document you do not need to specify a ledger since the total always goes to the appropriate control account in the GL.
account	String	The full (case sensitive) account code, within the ledger specified above, to which this total line is to be posted. For an accrual or prepayment type document you do not need to specify an account since the total always goes to the appropriate control account in the GL.
description	String	Optional description of up to 50 characters explaining what this total line is about. If you don't specify a description for this total line, any description in the header map will be used by default.
value	BigDecimal	The monetary value of this document excluding any tax. The sum of the 'value' fields from the line maps should equal this value here.
tax	BigDecimal	The monetary value of any 'tax' fields from the line maps. This is primarily a cross-check for validation purposes.

Key	Data Type	Comments
auto	Boolean	Optional and only applies to a total being posted to an AR or AP account. If set to true, an auto allocation will be performed on the account after any specific allocations (see below) have been performed.
allocations	List	Optional and only applies to a total being posted to an AR or AP account. A list of allocation maps to set this line off against other documents on the same supplier/customer account.

Allocation Map(s)

Any line (or total line) that is posted to a subsidiary ledger (AR or AP) account may optionally contain a list of allocations to specify the documents and values that this document is to be set off against. Typically, credit notes and cash are set off against invoices, but this need not always be the case. The monetary value in an allocation map is specified in its 'normal' form. For example, if the total line of a 100.00 purchase credit note was being fully allocated against two purchase invoices then the sum of the two allocation map values for the credit note would be 100.00. Anything less than 100.00 would mean that the credit note was not fully allocated. Anything above 100.00 would mean that the credit note was over allocated. Both under and over allocating is permitted. Each allocation map contains the following fields:

Key	Data Type	Comments
type	String	The document type to allocate against (e.g. 'SI' or 'ESI' etc.)
code	String	The document code to allocate against. The combination of the document type and document code uniquely identify the document being allocated against.
value	BigDecimal	The monetary amount of the parent line value to allocate against the above document.

Points of Note

Dates are always passed as Strings in the format 'yyyy-MM-dd' since, within the bookkeeping system, dates are independent of time zones. If you allowed the JSON converter to automatically convert a date it would be time zone sensitive which is not applicable for accounting documents (i.e. the date on a purchase invoice does not change if you mail it across the international date line!) Dates are checked for reasonableness (i.e. within plus or minus one year of 'today').

All monetary values within the accounting system are held as BigDecimal values. When transferring data, JSON uses Double values. Certain values simply cannot be accurately represented using floating point notation and you therefore run the risk of getting rounding errors. Consequently, you may pass all monetary values as Strings, if you wish, using code such as myBigDecimalValue.toPlainString() which, on recipt at the server, will be converted back to a

BigDecimal using: new BigDecimal(stringValueReceived). The server checks that the value you have submitted is valid for the currency of the document and will return an error if not (e.g. if you pass a US Dollar value with other than zero, one or two decimal places etc.)

The request to the server and the associated response from the server are all specified to use 'text/plain' MIME type rather than 'text/json' – even though the actual body of the request and response is JSON. This is done to stop Groovy and/or Grails from 'helping us to death!' The automatic conversion to/from JSON that Groovy and Grails are both capable of performing unfortunately 'consumes' the body of the request/response – but we need the raw bytes of the body to perform our security checking that the message has not been tampered with.

In a 'real' system, all RESTful requests and responses should be made over a secure https connection. However, for the sake of simplicity and ease of testing, the TLC system as supplied uses plain http for its RESTful web services.

Each request to a TLC RESTful service has a time stamp that is the number of milliseconds since the epoch (i.e. since 1/1/70). Only requests within 15 minutes either side of the server's current time will be accepted. This means that the clock on your remote client computer needs to be reasonably well in sync with the TLC server.

A document type within your company does not have to be flagged as allowing auto-generation of sequence 'numbers' in order to be used by the RESTful interface. By simply using the chosen document type within a REST call, the system assumes that it is to auto-generate the next sequence number for that document type.

A system supervisor can disable new logins/registrations and may also disable all further actions by logged in users by altering the operational state of the system. If the system supervisor disables just new logins/registrations this will *not* stop the RESTful interface. However, if the system supervisor chooses to also disable all further actions by users, this *will* stop the RESTful interface until the operating state of the system is changed back again.

Application Set Up

In the TLC application a user can be a member of any number of companies. The specific instance of a user and a company combination is called a Company User. The actions a Company User can perform are controlled by roles that you assign to the Company User. The data they can access is controlled by their membership of Access Groups.

Any Company User can also be given one or more Agent Credentials. Agent Credentials consist of a 20 character code, a 40 character secret and a status flag that specifies whether the credentials are active or not. When a remote application wishes to request a RESTful service of TLC, it must pass in the request header a time stamp, an agent code and a signature (HMAC SHA1) derived from the secret for the specified agent code. As long as the 'secret' is only known to the TLC server and the remote client, TLC is able to verify that the request is authentic and the time stamp allows TLC to confirm that this is not a duplicate request re-submitted at some later date.

To allow a client program, such as the test client described below, to access the system you need to choose the Company Administration → Security → Users menu option and then click on the Agent

Credentials arrow for the user who you would to give agent credential(s) to. You may then add new credentials for the user and set them to Active so that the test client can use them.

Test Client

A command line Groovy program called RestTest.groovy is available in the web-app/documentation directory of the tlc Grails project and contains a number of examples of how to post various documents to the TLC bookkeeping engine. Note that the program requires you to be connected to the Internet so that it can automatically 'grab' its dependencies. You also need to be using Groovy version 2.0 or better.

Once you have located the RestTest.groovy program, you will need to edit the tlcAgent and tlcSecret values near the top of the program to match the agent code and secret settings you have created for a Company User (see the Application Set Up section above). The tlcURL setting in the test program is initially set to issue requests to your local development server. If you wish to run tests across the Internet/Intranet to the some other TLC demo server, change the setting of the tlcURL as described in the comment within the program.

> **Note**: When you run your tests against a TLC demo server, there is a 'throttle' built in to the system that allows only one RESTful request per minute per company. This is a precaution against denial of service attacks. There is, of course, no such limitation for a live system.

Since the RestTest program is only an example of how to create RESTful client applications, its test data is hard coded in to the application in a property called tlcData. You will need to review and change this data as appropriate each time you run the application. See the Data Structure sections above for more information on how to create test data.

