ITAS API V2 INFORMATION FOR ANALYSTS AND DEVELOPERS.

Last updated: 7th July 2016

The purpose of this document is to introduce the ITAS API v2 Web Services. The intended audience is anyone involved in integrating with the ITAS API and anyone wishing to understand the relationship between the API endpoints and ITAS.

TABLE OF CONTENTS

ITAS API OVERVIEW

The purpose of the ITAS API is to provide external access to the ITAS system, both in terms of data and functionality. The API endpoints provide that point of contact by acting as a layer between ITAS and external systems. The ITAS API is a re-usable set of ITAS objects for both point-to-point communication between systems (i.e. no human intervention) and new user interfaces (e.g. web pages, mobile apps and so on). These systems can be built on top of the ITAS API by either by Hivedome or by clients themselves.

Terminology & Syntax

- The terms Web Service, Endpoint and API are used interchangeably and refer to a single http request that performs a purpose. An example of a web service is: http://{servername}/ReferenceData/{TradingEntityId}/Physicals/Ports/. This service delivers an array (list) of ports for the given Trading Entity.
- API (application programming interface) and the term ITAS API refers to all endpoints as a collective.
- Words in curly braces {servername}, {TradingEntityId} need to be replaced with real values in order to work i.e. TradingEntityId = AB
- V1 represents the previous version of ITAS Web Services that are due to be deprecated.
- V2 is the current version of ITAS Web Services now referred to as ITAS API.

Development History

With Web Services V1, Hivedome built an infrastructure to support complex communications but with limited functionality. It did allow development of reusable tools such as the Events Engine and State Machine which will migrated to V2 when V1 is deprecated.

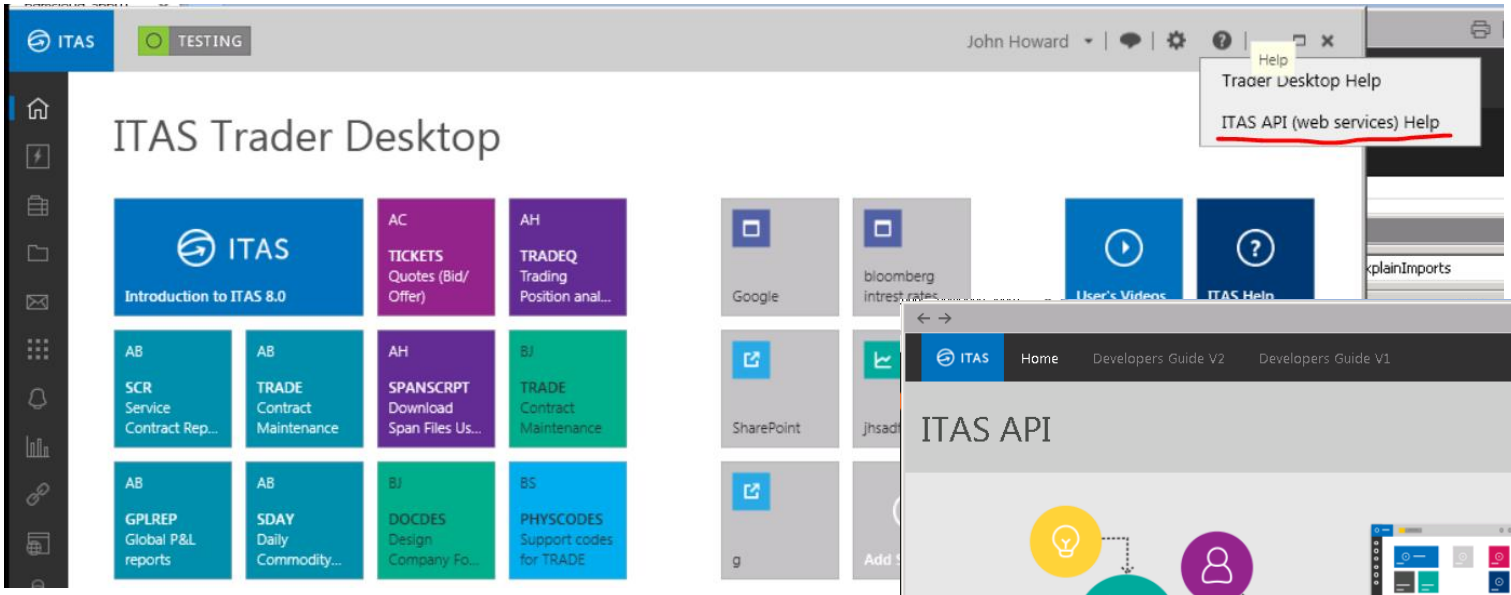
Help documentation

The API design is one thing; explaining how to use it is another. During the development of V1, definitions of the data to be exposed/consumed were published to a custom help portal and deployed to client sites and are accessible in your environment at http://{ApplicationServerName}:81. However feedback from API developers suggested the requirements went further than contract definitions. Requests for code snippets in multiple languages, data structure, object definitions, mock data examples and so on were needed.

Following consultation, Hivedome have refined the process of delivering documentation of the API endpoints for V2. A commonly use Web Services tool known as APIARY (https://apiary.io/) has been adopted. Apiary is an online API resource that enables all of the above. It provides a pseudocode that allows the API endpoints to be described to the extent that they can be dynamically mocked up quickly in any language. Being online this provides clients with a one-stop shop for documentation and investigation without writing a line of code.

Links to ITAS API help portals (V1 and V2) and Apiary are available from ITAS servers and can be found in two ways:

1. http://{servername}:4720 this link will only be available in each client's local network.
2. Via the Help Menu link in Trader Desktop



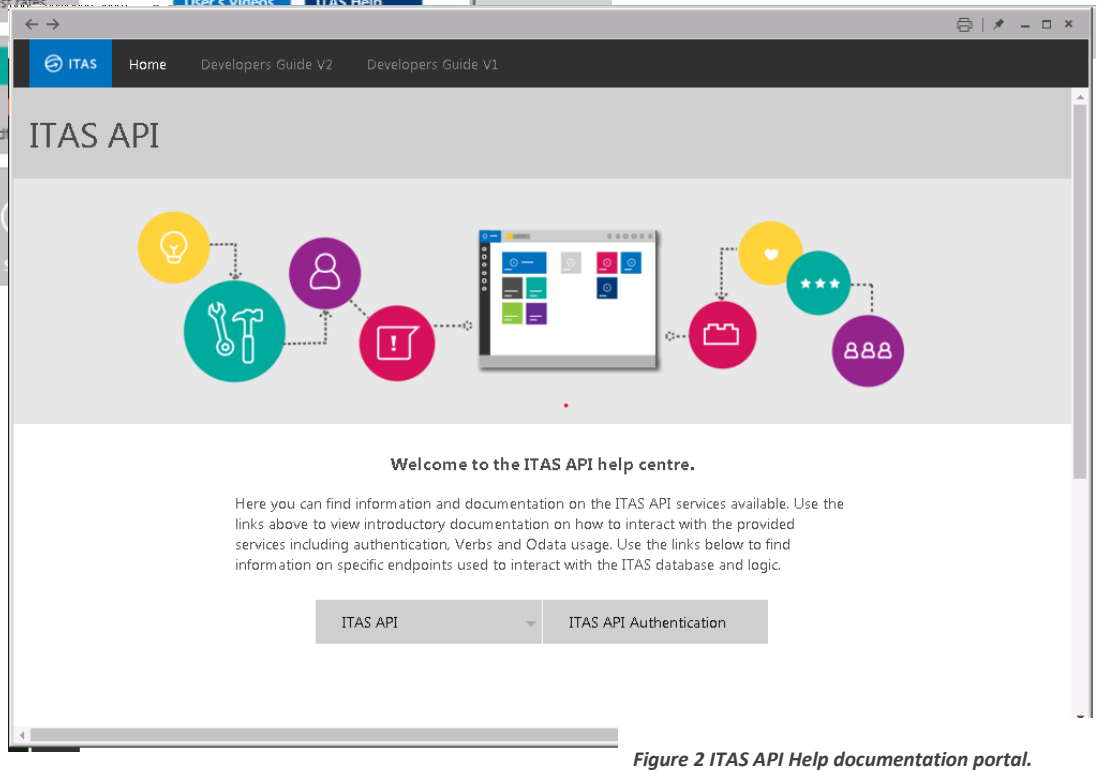


Figure 1 Link to ITAS API help pages.

The ITAS API Help Centre comprise of a variety of online resources such as:
** V2 Web Services Apiary (covering all endpoints),*
** ITAS Event Engine, * ITAS State Machine,*
** Developer Guides for V1 services and*
** Developer Guide for V2 services (this document).*

Figure 2 ITAS API Help documentation portal.

REQUESTS

With ITAS API V2, Hivedome have defined a model from scratch that will more accurately reflects the ITAS system as a whole. As a rule there is relationship between a single endpoint and a particular ITAS database table (e.g. /ReferenceData/{TradingEntityId}/ClientLedger/ClientAccounts/ClientAccountId exposes data from the xx_chr20 table), but this cannot be taken verbatim as the APIs also include business logic which may or may not read or update data in other tables as well.

Verb Usage

The ITAS API uses a set of standard verbs, common to many API platforms and attaches the focus on the purpose of the end point:

- GET (Read) – reads data from ITAS as a single object e.g. Physical Contract or Client Account.
- GET (List) - reads an array of data, i.e. multiple objects from the ITAS database.
- POST – creates new records in ITAS where ITAS determines and returns the ID e.g. a PayableOnAccount generic cash payment.
- PUT – creates or updates an object in ITAS where the ID is determined by the requestor e.g. ClientAccount
- PATCH – updates an object in ITAS with partial data; i.e. not all mandatory fields are sent in the request.
- DELETE – removes records from ITAS.

Important note: when records are updated via a PUT request, blank values are taken on face value and will overwrite data. For example to change the status of a chr20 record a PUT request made to /ReferenceData/{TradingEntityId}/ClientLedger/ClientAccounts/{ClientAccountId} with a body including:

```
{
"Status": "",
"CanBeBuyer": true
}
```

the Status field would be overwritten with a Null. So null values are never inserted by mistake, *only include values to be updated* in PUT requests, i.e.

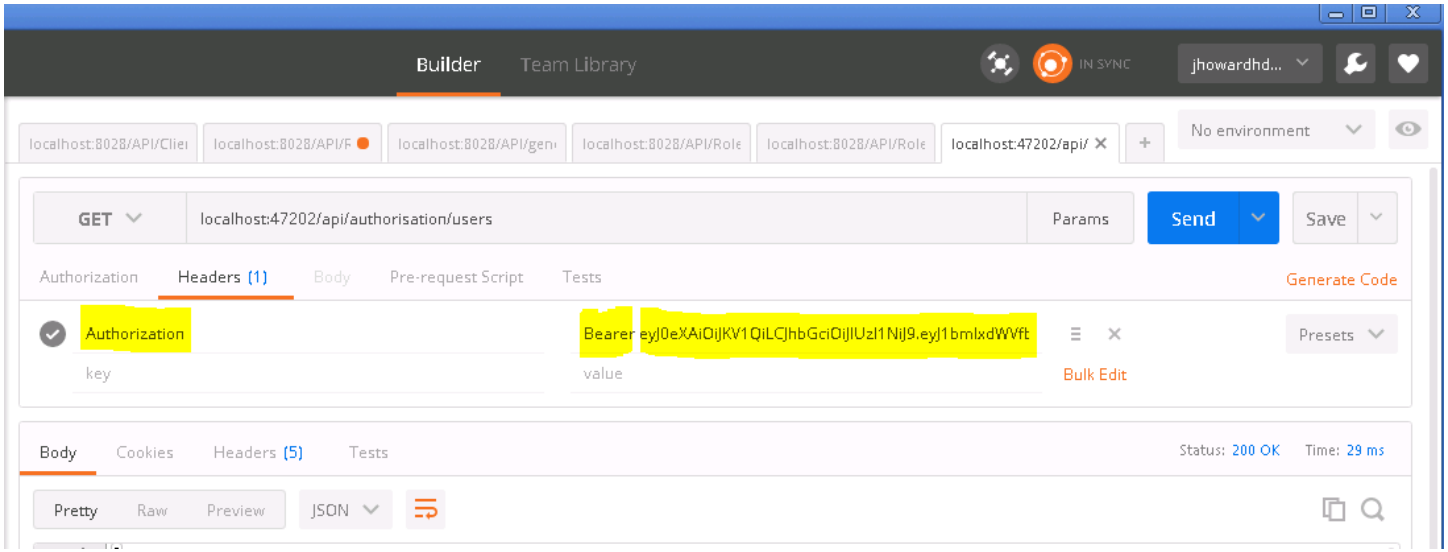
```
{
"CanBeBuyer": true
}
```

When a partial update is required, which does not include all mandatory fields, a PATCH request should be used.

Security

ITAS API V2 includes the introduction of a new security package using the OWIN open source specification and Katana components, both built and released by Microsoft. This approach is secure as tokens are not held in config; will also enable Role based authentication to be introduced.

Applying OWIN security is fairly simple as clients just need to include a header called `Authorization` with a value of "`bearer` " plus the token provided by the Hivedome support team during registration. Here is an example shown in the Chrome extension Postman



OData support

OData defines a system of querystring options that can help construct complicated queries to get the resources you want. As an architecture that's built on top of the current features of the Web, ITAS APIs support OData query strings. OData (Open Data Protocol) is an OASIS standard that defines best practice for building and consuming RESTful APIs.

When querying a collection (array or list) of, for example Nominal Accounts, the collection is specified through a URL. Query operations such as `filter`, `sort`, `paging` and `projection` can be specified as querystring options. The names of all system query options are prefixed with a dollar ($) character. The Querystring is proceeded with a question mark (?`) and each separated with an ampersand (&) sign.

For example, to page a result set the `$skip` and `$top` querystring parameters could be included as follows:

```
/ReferenceData/UM/NominalLedger/NominalAccount/?$skip=50&$top=20
```

This request returns items 51 to 70 in the list of nominal accounts.

OData Parameters

The full list of querystring parameters are:

Parameter	Durpose	Example
$filter*	The $filter system query option restricts the set of items returned.	Return all NominalAccounts whose DetailDescription does not contain the word 'HEADER' `$filter=substringof('HEADER', DetailDescription) eq false`
$orderby	Specifies an expression for determining what values are used to order the returned collection.	OrderBy values in the DetailDescription field in descending order. `$orderby=DetailDescription desc`
$skip	Returns a subset of entries (starting with Entry N+1).	Return items from entity 51 onwards. `$skip=50`
$top	Returns only the first N items of the set.	Return the first 20 `$top=20`

* See below for more details on this very powerful filter.

The real power of these filters come when combining the above parameters. Note that combinations are separated with the & character. For Example: return all NominalAccounts who's DetailDescription does not contain 'Header', in descending order and only return items 301 to 400.

```
/ReferenceData/UM/NominalLedger/NominalAccount/
?$filter=substringof('HEADER', DetailDescription) eq false&$orderby=DetailDescription desc&$skip=300&$top=100
```

The $Filter Parameter

A URI with a `$filter` query option identifies a subset of the Entries from the collection of entries identified by the Resource Path section of the URI. The subset is determined by selecting only the entries that satisfy the predicate expression specified by the query option.

The expression language that is used in `$filter` operators supports references to properties and literals. The literal values can be `strings` enclosed in single quotes, `numbers` and `boolean` values (true or false) or any of the additional literal representations shown in the Abstract Type System section.

The operators supported in the expression language are shown in the following table.

Operator	Description	Example
Logical Operators		
Eq	Equal	`/Suppliers?$filter=Address/City eq 'Redmond'`
Ne	Not equal	`/Suppliers?$filter=Address/City ne 'London'`
Gt	Greater than	`/Products?$filter=Price gt 20`
Ge	Greater than or equal	`/Products?$filter=Price ge 10`
Lt	Less than	`/Products?$filter=Price lt 20`
Le	Less than or equal	`/Products?$filter=Price le 100`
And	Logical and	`/Products?$filter=Price le 200 and Price gt 3.5`
Or	Logical or	`/Products?$filter=Price le 3.5 or Price gt 200`
Not	Logical negation	`/Products?$filter=not endswith(Description,'milk')`
Arithmetic Operators		
Add	Addition	`/Products?$filter=Price add 5 gt 10`
Sub	Subtraction	`/Products?$filter=Price sub 5 gt 10`
Mul	Multiplication	`/Products?$filter=Price mul 2 gt 2000`
Div	Division	`/Products?$filter=Price div 2 gt 4`
Mod	Modulo	`/Products?$filter=Price mod 2 eq 0`
Grouping Operators		
( )	Precedence grouping	`/Products?$filter=(Price sub 5) gt 10`

In addition to operators, a set of functions are also defined for use with the filter query string operator. The following table lists the available functions. Note: ISNULL or COALESCE operators are not defined. Instead, there is a null literal which can be used in comparisons.

Function	Example
String Functions	
bool substringof(string po, string p1)	`?$filter=substringof('Alfreds', CompanyName) eq true`
bool endswith(string p0, string p1)	`?$filter=endswith(CompanyName, 'Futterkiste') eq true`
bool startswith(string p0, string p1)	`?$filter=startswith(CompanyName, 'Alfr') eq true`
int length(string p0)	`?$filter=length(CompanyName) eq 19`
int indexof(string p0, string p1)	`?$filter=indexof(CompanyName, 'lfreds') eq 1`
string replace(string p0, string find, string replace)	`?$filter=replace(CompanyName, ' ', '') eq 'AlfredsFutterkiste'`
string substring(string p0, int pos)	`?$filter=substring(CompanyName, 1) eq 'lfreds Futterkiste'`
string substring(string p0, int pos, int length)	`?$filter=substring(CompanyName, 1, 2) eq 'lf'`
string tolower(string p0)	`?$filter=tolower(CompanyName) eq 'alfreds futterkiste'`
string toupper(string p0)	`?$filter=toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'`
string trim(string p0)	`?$filter=trim(CompanyName) eq 'Alfreds Futterkiste'`
string concat(string p0, string p1)	`?$filter=concat(concat(City, ', '), Country) eq 'Berlin, Germany'`
Date Functions	
int day(DateTime p0)	`?$filter=day(BirthDate) eq 8`
int hour(DateTime p0)	`?$filter=hour(BirthDate) eq 0`
int minute(DateTime p0)	`?$filter=minute(BirthDate) eq 0`
int month(DateTime p0)	`?$filter=month(BirthDate) eq 12`
int second(DateTime p0)	`?$filter=second(BirthDate) eq 0`
int year(DateTime p0)	`?$filter=year(BirthDate) eq 1948`
Math Functions	
double round(double p0)	`?$filter=round(Freight) eq 32d`
decimal round(decimal p0)	`?$filter=round(Freight) eq 32`
double floor(double p0)	`?$filter=round(Freight) eq 32d`
decimal floor(decimal p0)	`?$filter=floor(Freight) eq 32`
double ceiling(double p0)	`?$filter=ceiling(Freight) eq 33d`
decimal ceiling(decimal p0)	`?$filter=floor(Freight) eq 33`
Type Functions	
bool IsOf(type p0)	`?$filter=isof('NorthwindModel.Order')`
bool IsOf(expression p0, type p1)	`?$filter=isof(ShipCountry, 'Edm.String')`

RESPONSES

The following responses are sent from ITAS API:

- 200 – OK
- 201 – OK, New resource created
- 400 – Bad Request – The request was invalid. The exact error should be explained in the response. E.g. Invalid Json, missing attributes
- 401 – Unauthorized – The request requires authentication
- 403 – Forbidden – The server understood the request, but is refusing it or the access is not allowed.
- 404 – Not found – The URI data is incorrect. For example the Trading Entity Id or object Id (eg DocRef) does not exist.
- 500 – Internal Server Error – This error is not returned by the application but by the server.

The ITAS Response Envelope

All responses from ITAS API V2 endpoints are contained within a response envelope. This is a new feature of V2 and does not exist in V1. The Envelope contains four parts, each part is only sent in the response if it populated:

- Data – used for the body data e.g. a list of entities or details of a specific entity, or the id of an entity after it has been created. This can be either an *array* or an *object* depending on the endpoint called.
- Response – used to denote the response type e.g. `success` or `failure`. Always a *string*.
- Pagination – when the response contains a list of entities, the pagination section includes meta-data associated with the list. *Object*.
- Messages – used to return error, warning and information messages. *Array*.

Example

```
{
   "data": [] or {},
   "response": "value",
   "pagination": {},
   "messages": []
}
```

Pagination

The Pagination section of the response envelope describes the dataset returned in the `Data` part of the response envelope:

- Total – total number of records in the dataset.
- Limit – how many records can be returned. The default is 200 and if there are more than 200 records in the available dataset only 200 will be returned by default. This can be overridden by adding a `Limit` value to the Odata querystring i.e. $limit=1000
- Offset – the position of the first record to be returned from the dataset. i.e. $offset=100 will return records starting at position 101.
- Returned – actual number of records returned.

The following paginatino describes the records returned as: 243 records, stating at position 1301, from a total or 1543 records with a limit of 300. Note that the reason only 243 records are returned when 300 are requested is that the dataset finishes before the 300'th record is found.

```
"pagination":
   {
       "total": 1543,
       "limit": 300,
       "offset": 1300,
       "returned": 243
   }
```

Success Response Examples

200 GET - Object

```
GET ReferenceData/Global/Currency/CurrencyId/aud


{
  "Data": {
      "CurrencyId": "AUD",
      "CurrencyDescription": "Austrailian Dollar"
    },
  "Response": {
    "Value": "Success"
  }
}
```

200 GET – List

```
GET /ReferenceData/Global/Country/?$skip=50&$top=2

{
  "Data": [
    {
      "CountryId": "CY",
      "CountryDescription": "Cyprus"
    },
    {
      "CountryId": "CZ",
      "CountryDescription": "Czech Republic"
    }
  ],
  "Pagination": {
    "Total": 222,
    "Top": 2,
    "Skip": 50,
    "Returned": 2
  },
  "Response": {
    "Value": "Success"
  }
}
```

200 PUT – Create (Id determined by the URI)

```
PUT /Accounting/AB/Transaction/TransactionHeader/CP1500020

{
  "Data": {
    "DocumentReference": "CP1500020"
  },
  "Response": {
    "Value": "Created"
  }
}
```

200 POST - Create (Id determined by ITAS)

```
POST /Finance/AB/Cash/PayableManual

{
  "Data": {
    "CashDocRef": "XX1234567"
  },
  "Response": {
    "Value": "Created"
  }
}
```

201 PUT – Update (full update i.e. includes all mandatory fields)

```
PUT /Accounting/AB/Transaction/TransactionHeader/CP1500020

{
  "Data": {
    "DocumentReference": "CP1500020"
  },
  "Response": {
    "Value": "Updated"
  }
}
```

201 PATCH – Delta update (partial update i.e. does not include all mandatory fields)

```
PUT /Accounting/AB/Transaction/TransactionHeader/CP1500020

{
  "Data": {
    "DocumentReference": "CP1500020"
  },
  "Response": {
    "Value": "Updated"
  }
}
```

200 DELETE - Delete (reserved for future use)

```
DELETE /Accounting/AB/Transaction/TransactionHeader/CP1500020

{
  "Data": {
    "DocumentReference": "CP1500020"
  },
  "Response": {
    "Value": "Deleted"
  }
}
```

Error Response Examples

Error responses are served as 400 or 404 Bad Request, the exact details can be found in the message payload. All responses follow the same format.

404 Not Found Examples

```
POST /Finance/AB/Cash/Payable_BADURL_Manual

{
  "Response": {
    "Value": "NotFound"
  },
  "Messages": [
    {
      "Severity": "Error",
      "Message": "No HTTP resource was found that matches the request URI 'http://Servername/Finance/um/Cash/
Payable_BADURL_Manual'."
    }
  ]
}
```

```
GET /ReferenceData/PO/WIPLedger/Charter/

{
  "Response": {
    "Value": "404 Not Found"
  },
  "Messages": [
    {
      "Severity": "Error",
      "Message": "API5067 - API : The value po sent as the TradingEntitiyId does not exist."
    }
  ]
}
```

400 Bad Request Examples

```
POST /Finance/AB/Cash/PayableManual (DocumentDate sent incorrectly in request body)

{
  "Response": {
    "Value": "BadRequest"
  },
  "Messages": [
    {
      "Severity": "Error",
      "Message": "API5060 - Data : The value sent for the field DocumentBody.DocumentDate is of the wrong data type"
    },
    {
      "Severity": "Error",
      "Message": "API5058 - Data : The mandatory field DocumentDate was missing in the request body."
    }
  ]
}
```

Message Codes

Messages are an array of messages, errors and warnings. The `Severity` field within each message can be used to determine the importance and is defined as:

- Error – Request aborted; serious problem with the request.
- Warning – Request processed but the way it has been structured may cause issues in a future request.
- Information – Request accepted but there may be a better way to achieve this.

Detailed `Message` section of each message are structured in the following format.

```
{ErrorID} – {Category} : {Message description}
```

For example

```
API5054 - API : More than one MasterData company is configured in the ITAS Heritage database which is not allowed. Operation aborted.
```

The table below shows a list of error messages and their codes.

Code	Full Description
API5054	API5054 - Config : More than one MasterData company is configured in the ITAS Heritage database which is not allowed. Please contact Hivedome support to check Masterdata configuration.
API5058	API5058 - Data : The mandatory field {fieldName} was missing in the request body.
API5059	API5059 - Data : The value sent for the field {fieldName} exceeded the maximum number of allowed characters
API5060	API5060 - Data : The value sent for the field {fieldName} is of the wrong data type
API5061	API5061 - Connection : Database connection error, please contact Hivedome Support to check configuration parameters.
API5062	API5062 - API : Body of the request has incorrect JSON format.
API5066	API5066 - API : Incorrect routing. There is an issue with the URI of the request made please check.
API5065	API5065 - API : Authorisation Error
API5064	API5064 - API : Token Expired
API5063	API5063 - API : Incorrect token role
API5067	API5067 - API : The value {TradingEntityId} sent as the TradingEntitiyId does not exist.
API5068	API5068 - Data : No records found.
API5069	API5069 - Data : The value send in the URI variable {PropertyValue} does not exist in the ITAS database for that Trading Entity.
API5070	API5070 - Data : Querystring variables {Property} and {PropertyValue} are dependant. Either send both or neither.
API5071	API5071 - Data : A value in the URI and the Body of the request do not match.

API5072	API5072 - Data : There was an error with the body of the request.
API5073	API5073 - Data : Entity {EntityValue} passed in field {FieldValue} was not found for TradingEnity {TradingEntityId}
API5074	API5074 - Data : The value sent in the URI variable {Property} is not allowed.
API5075	API5075 - API : The combination of querystring parameters used do not return any records. Please amend or remove the querystring.
API5076	API5076 - API : The entity {EntityId} already exists for Trading Entity {TEID}. Creating another with the same ID is not allowed.
API5077	API5077 - API : That method is not supported, please check the ITAS API documentation.

OTHER CONSIDERATIONS

Versioning

The intension is to extend the API through a series of iterations. Version 2 is the major version and will consist of endpoints around Reference Data as well as allowing us to rollout the new AccessToken security model.

Applying a similar version control to that adopted for ITAS 8, we will use the x.y.z format where:

x = major version (currently 2 for ITAS API)
y = minor version (will indicate the release package)
z = build number (internal identification)

In terms of APIARY there will be a 'version' per release – current release (2.5) can be seen here http://docs.itasapiv2r5.apiary.io/#

By creating separate 'version' per release, API developers will be able to test against a UAT API while supporting a different PROD version. When changes are released, updates will be highlighted within each release notes document available here http://itashelp.hdmng.com/release_notes.htm. The version deployed to each client site will be available to view within Trader Desktop.