ORACLE
INSURANCE

# Oracle Health Insurance Back Office

## HTTP Service Layer (HSL) User Manual

Version 1.15

Part number: F27899-01

September 1, 2020

# CHANGE HISTORY

| Release | Version | Changes |
|---|---|---|
| 10.16.2.3.0 | 0.1 | • New (internal) document CDO 15195<br>• Removed references to XML |
| 10.17.1.4.0 | 0.2 | • Revised document. |
| 10.17.1.4.0 | 0.3 | • Revised document<br>• Added chapter on 'Language Aspects' |
| 10.17.1.4.0 | 0.4 | • Processed feedback from review by EVI |
| 10.17.1.4.0 | 0.5 | • Processed feedback from review by KOS |
| 10.17.1.4.0 | 1.0 | • Some minor textual changes, increased to version nr 1.0 |
| 10.17.2.0.0 | 1.1 | • Revised URLs for retrieving Swagger definition<br>• Revised comments for PL/SQL client code fragment<br>• Revised 'Error Handling' paragraph.<br>• Added 'Input Validation' paragraph |
| 10.17.2.2.0 | 1.2 | • Revised HSL Technical Principles<br>• Added 'Basic Authentication' to Terminology |
| 10.18.1.0.0 | 1.3 | • Changed title<br>• Added 'security aspects' to introduction with reference to **Doc[1]** for OAUTH2 and OPTIONS support<br>• Revised security item in 'HSL technical principles' |
| w10.18.1.3.0 | 1.4 | • Updated HTTP return codes<br>• Added paragraph about optimistic locking |
| 10.18.1.4.0 | 1.5 | • Added 'How to bypass optimistic locking' to optimistic locking paragraph |
| 10.18.1.4.0 | 1.6 | • Updated HTTP Return Codes<br>• Updated 'Optimistic Locking' |
| 10.18.2.0.0 | 1.7 | • Design Principles – updated<br>• HSL Technical Principles – updated<br>• Error Handling – clarified text<br>• Language Aspects: removed remarks about domain-based enumerations<br>• Language Aspects: removed 'Intended Behaviour' paragraph |
| 10.18.2.2.0 | 1.8 | • Added specific usage aspects regarding HSL_C2B service operations. |
| 10.18.2.3.0 | 1.9 | • Some minor textual changes. |
| 10.19.1.0.0 | 1.10 | • No changes. Republished with different part nr. |
| 10.19.1.2.0 | 1.11 | • Documented x-ohibo-modification section in runtime swagger in the Service Definition structure paragraph. |
| 10.19.1.4.0 | 1.12 | • Error handling: operation of developermode changed |
| 10.19.2.0.0 | 1.13 | • Updated HTTP Return Codes and Optimistic locking |
| 10.20.1.0.0 | 1.14 | • No changes, republished. |
| 10.20.6.0.0 | 1.15 | • Interfacing Decisions slightly enhanced |

# RELATED DOCUMENTS

A reference in the text (doc[x]) is a reference to another document about a subject that is related to this document.
Below is a list of related documents:

**Doc[1]** OHI Back Office HTTP Service Layer (HSL) Installation & Configuration Manual (CTA13681)

**Doc[2]** OHI Back Office JET Application Installation & Configuration Manual (CTA13686)

# Contents

# 1 Introduction

The OHI Back Office HTTP Service Layer (HSL) is used to implement operations for so-called Use Case Services. These services support (parts of) typical processes relevant to OHI BO customers.

The end-user applications that will use these operations are likely to be used by self-service users (insured members, care providers etc) but also by call-center operators.

The HTTP Service Layer is based on RESTful services technology which has the following advantages for current web application frameworks (like AngularJS and Oracle JET):

- accessible through HTTP
- supports JSON as input and output formats
- standardized interface language through using HTTP verbs (GET, POST, PUT, PATCH, DELETE, OPTIONS)
- standardized set of exceptions through HTTP error codes

It is a strict security requirement not to expose the HSL service directly to the internet but hide the HSL service behind an intermediary REST service. This has the following advantages:

- end user security can be implemented in the intermediary REST service.
- support a central monitoring and security implementation.
- allows additional code and helps to bridge interface changes in subsequent versions of the OHI-supplied services in a central location, only once.

The functional implementation of the HSL services is done in PL/SQL.

A generated Java layer exposes the HSL services through the Weblogic Application Server as RESTful services.

Using PL/SQL as the basis for HSL service means that the HSL operations can also be accessed through PL/SQL within the OHI Back Office database. This may provide a performance benefit because it bypasses the overhead of the Weblogic Application Server and obviates the need to serialize and deserialize objects.

## 1.1 Security Aspects

Default authentication for the HSL services is Basic Authentication over SSL.

In 10.18.1.0.0 the following changes were made to the security of the HSL services:

- No authentication needed for OPTIONS method
  This requires that the HSL service is deployed with 'Custom Roles and Policies'
- OAUTH 2.0 token authentication and validation as an alternative to Basic Authentication

See **Doc[1]** for more information.

## 1.2 Purpose of This Document

This document describes the generic aspects of the HTTP Service Layer.

The functionality and interface of each service is described in a so-called Swagger schema which can be retrieved once the service is deployed to the application server.

A terminology list is included at the end of this document.

For information regarding installation and configuration of OHI Back Office HTTP service layer components please use **Doc[1]**.

# 2 Design Principles

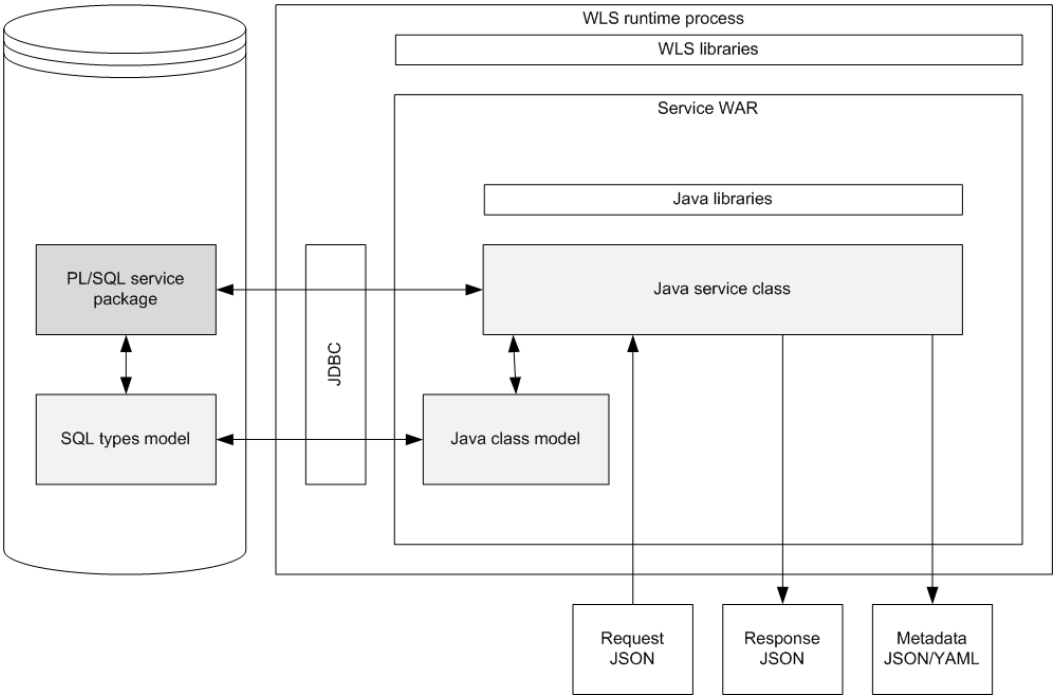
The HTTP Service Layer is based on RESTful service concepts and the following architecture decisions:

- PL/SQL is used to implement the functionality, using SQL types as content containers
- the HTTP interface layer is implemented in Java and exposed through WLS and PL/SQL
- a Java class model is used to pass data from SQL types and vice versa
- Java is used to generate metadata and schema information
- Java is used for technical validation of request data (datatype, presence of mandatory data etc).

To avoid confusion and prevent dependency with the existing SOAP-based SVL components, the HSL services will have their own SQL types and tooling.

Where possible, existing Java libraries and frameworks such as JPA-RS, Jersey, MOXy and JDBC are used to handle the interfacing.

The diagram below shows the various parts of a HSL service:




The components shown in grey are maintained by OHI BO.

## 2.1 HSL Interfacing Decisions

This paragraph describes the interfacing decisions taken by OHI Back Office which may affect how you write and test your client applications:

1. Terminology, terms and documentation will be in the English language.

   Rationale: the service layer is typically used by developers who are used to technical documentation in English.

2. A Use Case service is designed to perform a specific set of tasks.

   Rationale: this allows a more compact object model for each service and makes writing client applications more straightforward.

3. A HSL service operation typically applies on a *resource.*

   Rationale: The key abstraction of information in REST is a resource (dissertation of Fielding outlining RESTful principles).

   In OHI BO, a *resource* is an object which can be accessed through a service operation and which can (normally) be mapped on a OHI BO entity such as a policy, claim, person etc.
   A *collection resource* is a list of *singular resources.* Example: a *collection resource* `claimCollection` is a list of `claim` resources.

4. The functionality of HSL services is implemented in PL/SQL.

   Rationale: implementing the functionality in the database is more efficient and saves many roundtrips compared to implementing application functionality in the middle tier.

5. The operations of a HSL application may be accessed through PL/SQL and Weblogic Application server.

   Rationale: allows PL/SQL to access the same functionality as the HTTP interface (although its use may be different).

6. The interface definition is aligned with Oracle internal standards.

   Examples: resource concepts, HATEOAS links, pagination, metadata.

   Rationale: align with internal Oracle requirements (most of which also improve integration facilities right out of the box).

7. Each service is based on a Swagger 2.0 schema.

   Rationale: apart from an internal requirement within Oracle, Swagger 2.0 specifications provide a readable and detailed interface description which can be used by analysts, programmers and testers.

8. Each service has its own objects.
   Apart from common definitions for (primitive) scalar types, no objects are shared between services.

   Rationale: since the changes to a class model of a service are local to that service, these changes have no impact on other services.

9. Inbound and outbound resources may be in JSON format.

   Rationale: required by Oracle internal standards.
   JSON (Javascript Object Notation) has become the de facto standard for

serializing objects in Internet applications.

Note that XML format is not supported.

10. POST/PUT operations assume that the inbound resource is a complete object.

    When validating the inbound object, an exception will be raised if one of the required fields is not present.

    Rationale: consistent with HTTP convention.

11. PATCH operations assume that the inbound resource may be an incomplete object.

    When processing the inbound object, only those fields which are set are validated.

    Rationale: consistent with HTTP convention.

12. ~~Domain-based enumerations are mapped to domain meanings if Accept-Language is set.~~

13. The Java layer provides basic validation for parameters and inbound objects

    Rationale: provide feedback to the calling application as soon as possible. Note that domain-based enumeration values are validated in the PL/SQL layer.

    Rationale: avoid calling the database back end unnecessarily.

14. Path templating is used to clarify the meaning of parameters.

    Example of a template path : /v1/relation/**{rel_nr}**

    Rationale: allows various fields to identify resources and sub resources.

15. Each resource URI must have a version number.

    The initial version of a resource is v1.

    Rationale: required by Oracle internal standards.

16. The HSL objects are separate from the SVL objects.

    Rationale: avoid interdependency with SVL services.

17. The HSL services are designed for application to application interface and should never be exposed for direct calls in some form of user environment. Any form of authentication, authorization or data access is always the responsibility of the calling application.

    Rationale: To prevent any incompatibility with any form of authentication or authorization of the calling application the services can be used for many interface scenario's, to support for example the development of portal applications or to develop an automated interface with a third party.

## 2.2 HSL Technical Principles

The following technical principles may be of interest for client application developers and administrators:

1. WebLogic Server will be used as the standard application server deployment platform.

2. Default security is basic authentication over SSL, requiring the client to login as a Weblogic user 'restuser' or self defined users (depending on the deployment option used). Basic authentication is also required for retrieving the Swagger definition from the deployed HSL application.
   OAUTH2 is supported starting with the 10.18.1.0.0 release.
   See **Doc[1]** for more information.

3. The service calls will be stateless, which implies that subsequent service calls are not aware of each other.

   Rationale: characteristic of all HTTP applications.

4. A single service operation will contain one or more atomic transactions.

   Rationale: a service operation should be designed to leave the database in a consistent state. If a single operation is implemented through multiple transactions, each transaction should leave the database in a consistent state.

5. Locking is kept to a minimum.

   Rationale: HSL service operations are stateless and designed for short running transactions.
   Records will be locked with the nowait option immediately before being updated. If a record cannot be locked, an error message is returned and the transaction is rolled back.
   Unless specifically stated, 'optimistic locking' is not used, ie. the contents of the resource are not checked with a previous known state to decide whether the update of a resource can be applied.

6. Standard HTTP return codes are used.

   Rationale: standard practice in line with Oracle internal standards.

7. Functional faults will support language dependent error messages.

   Rationale: although the services are in English the functional error messages returned will use the multi-language support as present in the OHI Back Office application; this to be able to return language specific messages based on the calling context.

8. Standard Java logging is used for error, informative and debug level log messages.

   Rationale: by adhering to a common standard logging mechanism this will be easier to configure and use for system administrators who are experienced with Java based application management.

9. HSL services are synchronous services.

   Rationale: at this stage, asynchronous services are not in scope.

10. Additional technical requirements will be implemented as much as possible through applying standard solutions that are compatible with Weblogic Server.

    Rationale: focus on delivering functionality and making use of standard components instead of developing proprietary solutions.

11. Date and date time values are assumed to be in local time.

    Rationale: OHI Back Office currently supports local time only.

# 3 Generic usage aspects of HSL services

Due to the high level of standardization of HSL services the interface of each HSL service is much the same.

The remainder of this document describes the generic aspects of HSL services.

- HTTP requests and HTTP return codes
- Retrieving the service definition
- Retrieving configuration information
- singular resource operations
- collection resource operations

## 3.1 Creating HTTP requests

It is assumed that you know how to create HTTP requests to call HSL service operations.

If you are testing, there are many ways to send HTTP requests to a HSL application. You may want to use the Google Chrome app Postman or SoapUI. In our examples we will use curl to send a HTTP request to the HSL application.

### 3.1.1 HTTP Request Headers

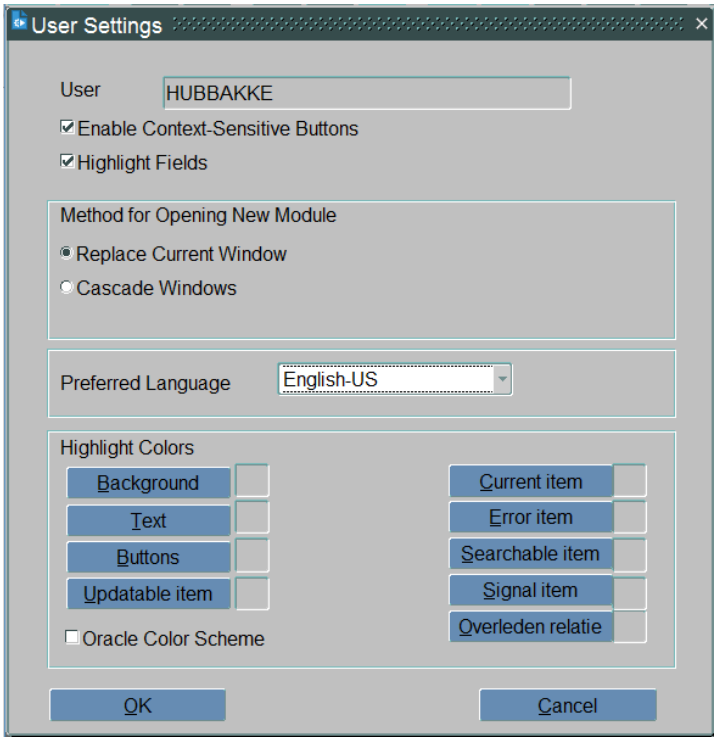
For incoming requests, the HTTP request headers are tested as indicated in the following table:

| Request Header | Optional? | Regular expression | Example value |
|---|---|---|---|
| **Accept** | N | ^application/json( *; *charset=utf-8)* | application/json |
| **Accept-Language** | Y | ^[a-z][a-z]-[A-Z][A-Z]$ | nl-NL |
| **Content-Type** | Y | ^application/json( *; *charset=utf-8)* | application/json; charset=utf-8 |
| **Content-Length** | Y | [0-9]+ | 123 |

Note: if the request is for /api/swagger.yaml, the value for the 'Accept' header should be 'application/yaml'.

### 3.1.2 Accept-Language

The default session language is the preferred language as set within the OHI User Settings window for the OHI officer.
The OHI officer (and by implication, the session language) is set through the call-context.




The session language can be overruled through the HTTP request header 'Accept-Language'.

Example values:

- nl-NL
  language: Dutch, country: Netherlands

- en-US
  language: English, country: United States

Notes:

- the country (territory) for all OHI BO customers except PlanSeguro is always 'NL'

- if you are creating requests directly from a browser, the browser may add the 'Accept-Language' header to the request without your knowledge.

- If the OHI officer is set from within the internal OHI application code after processing the call context, the previously set session language is overruled.

## 3.2 HTTP Verbs

The following HTTP verbs may be used:

- GET
  Retrieve the resource located at the URI.
  This is the default verb.
  Idempotent operation: a subsequent call will return the same contents for the resource if the underlying data has not been changed since the previous invocation.

- POST
  Create new data.
  A subsequent call with the same data will return an error.
  Idempotent operation: a subsequent call with the same data will NOT result in the creation of a new resource.

- DELETE
  Delete the resource located at the URI.
  A subsequent call with the same data will return an error.
  Idempotent operation: a subsequent call will not delete any data if the first call already deleted the data corresponding with the resource.

- PUT
  Used to replace the contents of the resource located at the URI.
  Data which is absent from the inbound resource will be deleted from the OHI Back Office database.
  Idempotent operation: a subsequent call with the same data will have the same effect as the first call.

- PATCH
  Process the contents of the resource located at the URI to update the underlying OHI BO data.
  Data which is absent from the resource is not processed.
  Idempotent operation: a subsequent call with the same data will have the same effect as the first call.

## 3.3 HTTP Return Codes

Typical HTTP return codes for HSL requests:

- 200 (OK)
  Request successfully processed.

- 201 (CREATED)
  Will be returned for a POST request.

- 400 (BAD_REQUEST)
  May be returned if parameter validation failed at the Java level.

- 404 (NOT_FOUND)
  Will be returned in the following situations:

  - Requested URI was not recognized.
  - Parameter validation failed at the PL/SQL level.
  - OHI Back Office data could not be found.
  - Parameter validation failed at the Java level
    (the currently used version of the Jersey library returns HTTP 404 when parameter validation fails).

- 405 (METHOD_NOT_ALLOWED)
  The HTTP verb was not recognized.

- 406 (NOT ACCEPTABLE)
  Should occur if a non-existing representation is required. For example if 'Accept' is set to 'application/xml'.

- 412 (PRECONDITION_FAILED)
  Data already updated by another user.

- 422 (FUNCTIONAL ERROR)
  OHI BO business rules were violated.
  An error message is returned – see note below

- 423 (LOCKED)
  Record is locked by another user.

- 500 (INTERNAL_SERVER_ERROR)
  This is a catch all for failed requests that were passed to the HSL service. Possible causes are:

  - database connection not working

  - missing or invalid PL/SQL components.

Note:

- Error messages are written to the service log file.

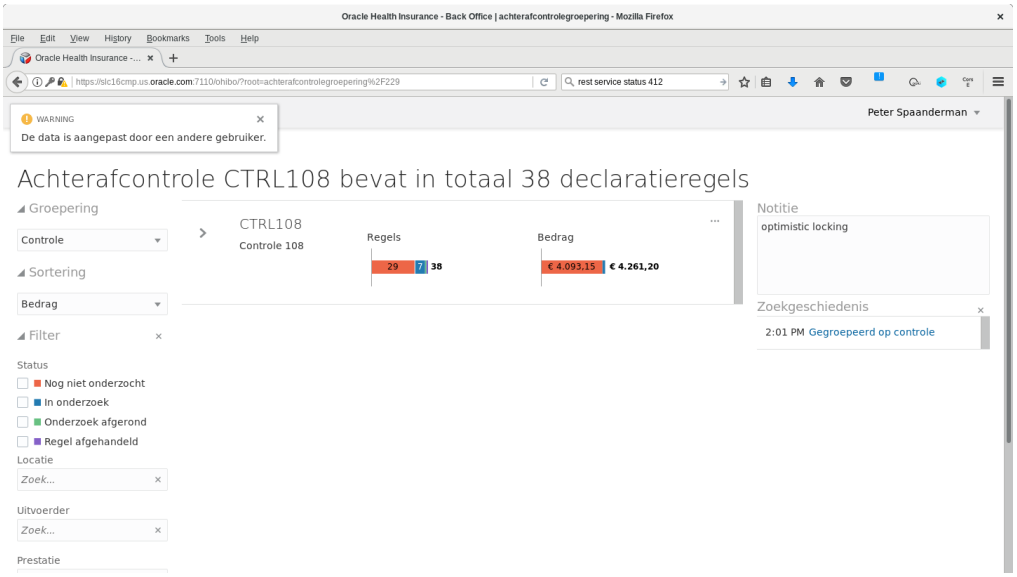
- For security reasons, technical error messages are suppressed from the response by default. Technical error messages are only included in the response if the application is run in developer mode. Functional error messages (HTTP 412, HTTP 422, HTTP 423 and HTTP 400) will never be suppressed.

## 3.4 Optimistic Locking

As from release 10.18.1.3, OHI BO has started the implementation of optimistic locking for HSL (and PSL) web services.

The purpose of optimistic locking is to abort an update transaction and return HTTP 412 (PRECONDITION_REQUIRED) if the data were already updated by another user. If the record is locked by another user, HTTP 423 (LOCKED) is returned. Note that HTTP 412 was previously used for functional errors as well.

The screenshot below demonstrates how a failed update resulting in HTTP 412 might be signalled to the user:




Implementation of optimistic locking as per release 10.18.1.4:

- Optimistic locking has been implemented in PSL_ACL.
  HTTP 422: functional error
  HTTP 412: data updated by another user

- Optimistic locking has NOT yet been implemented in HSL_REL and HSL_POL
  HTTP 412: precondition failed (functional error)

### 3.4.1 How to bypass optimistic locking

Optimistic locking is the default, but sometimes the update must be forced anyway.

If this is desired, strip the md5 value in the resource that should be updated before calling the PUT or PATCH operation.

## 3.5 Input Validation

At the Java level, input parameters are validated as follows:

- Parameters and object members of the Date type are converted from a string value (yyyy-mm-dd) to a date value.

- Parameters and object members of numerical types are converted from a string value to an integer or BigDecimal value.

- Parameters and object members of the String type are matched with a regular expression.

- Parameters and object members of an enumeration type are matched with the allowed values for that enumeration type.

The validation fails in the following cases:

- A data conversion error

- a string value does not match with its predefined regular expression.

- an enumeration value does not match with its allowed values.
- a value is not within its designated minimum-maximum range.
- a missing value for a NotNull parameter or object member

## 3.6 Error Handling

The implementation of Use Case Services uses PL/SQL to access and manipulate data. While processing the request, the OHI Back Office business rules come into play and raise exceptions if your data is incomplete or incorrect.

Check the functional specification or the online help in the OHI Back Office Forms GUI application ('Help → Inhoud en Index → Use Case Services') for a list of possible errors for a given HSL service.

Our advice for validating a client application based on Use Case Services is to always include various tests with new data.

### 3.6.1 Developer mode and Production mode

While developing a client application, meaningful error messages help to understand how correctly invoke the HSL service.
If you set the hsl.properties property

```
hsl.<application>.developermode=true
```

none of the error messages in the responses for that service will be suppressed.

If you set the hsl.properties property

```
hsl.<application>.developermode=false
```

only functional errors (Bad request and business rule violations) will be disclosed, because these are needed by the user of the application to revise input mistakes. All other errors (e.g. internal server errors) will be suppressed.

Since meaningful error messages in production systems are a security risk, 'developer mode' is turned off by default.

The following example illustrates the differences in error handling between developer mode and production mode.

### 3.6.2 Developer mode

The following helpful response may be generated when running in developer mode, indicating a programming issue:

```
{
"attribute":"getRelationByNumber.xpand",
"internalStatus":"Internal Server Error",
"invalidValue":"23424987897skjdfjkhkjhjk238798798sdukykjy345987
egfuiyiuyui3456uiyuiy3ui6yui3y4ui5yuiy34ui5yui34y5uiy345uiy3453
4uyyuiyuiuiiu",
"message":"divisor is equal to zero"
}
```

By default, the message is suppressed:

```
{
"attribute":"Undisclosed",
"internalStatus":"Internal Server Error",
"invalidValue":"Undisclosed",
"message":"Undisclosed"
}
```


In order to see what caused the problem we should look at the log:

```
Dec 06, 2017 10:38:23 AM
com.oracle.insurance.ohibo.exception.ExceptionResponse
setMessage
SEVERE: message: divisor is equal to zero
Dec 06, 2017 10:38:23 AM
com.oracle.insurance.ohibo.exception.ExceptionResponse
setAttribute
SEVERE: attribute: getRelationByNumber.xpand
Dec 06, 2017 10:38:23 AM
com.oracle.insurance.ohibo.exception.ExceptionResponse
setInvalidValue
SEVERE: invalidValue:
23424987897skjdfjkhkjhjk238798798sdukykjy345987egfuiyiuyui3456u
iyuiy3ui6yui3y4ui5yuiy34ui5yui34y5uiy345uiy34534uyyuiyuiuiiufas
df
```

Since the log cannot be read by outsiders, the risk of malicious use by hackers is reduced.

# 4 Service Definition

The functionality of Use Case Services is documented in the online help of the OHI Back Office GUI (Help → Use Case webservices). This information is derived from the functional specification and maintained manually.

The service definition for HSL services is a Swagger 2.0 schema generated by the deployed HSL application.

The Swagger definition documents both the operations and the objects used by a service. Swagger (new name: 'OpenAPI') is a definition standard (http://swagger.io) supported by many leading software vendors including Oracle.

The Swagger definition of each HSL application provides useful documentation to client application developers. Swagger definitions can also be used as the basis for code generation.

The Swagger definition is exposed as follows:

- https://server:port/application/api/swagger.json
  Returns the Swagger definition in JSON format
- https://server:port/application/api/swagger
  Returns the Swagger definition in JSON format
- https://server:port/application/api/swagger.yaml
  Returns the Swagger definition in YAML format

The URI to expose the Swagger definition of the POL service would look like this:

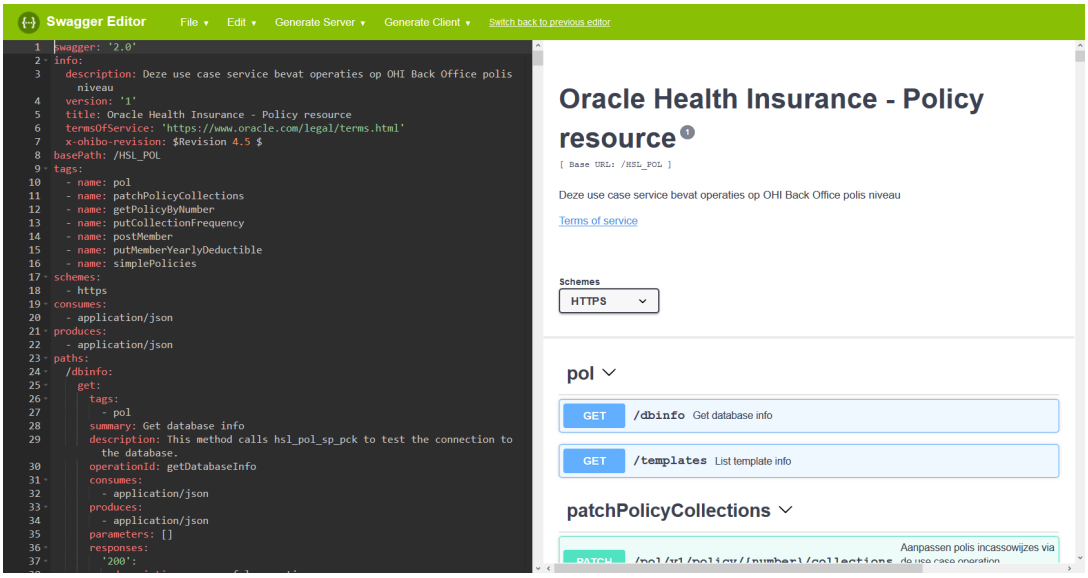
https://localhost:7001/HSL_POL/api/swagger.json

## 4.1 Viewing the Swagger Definition

The online Swagger editor (http://editor.swagger.io) provides a user friendly user interface to browse the Swagger definition.

In the following example we use the online Swagger editor to view the Swagger definition of the POL service:

- Browse https://server:port/HSL_POL/api/swagger.json
- Copy the contents of the browser window
- Start the Swagger editor
- Choose 'File > Paste Json' and paste the contents of the browser window into the edit buffer.




## 4.2 Understanding Swagger

In case you are not familiar with Swagger these links will get you started:

- http://swagger.io/specification/
  Contains the Swagger specifications

- http://petstore.swagger.io/
  A simple example to learn how the Swagger specification hangs together.

## 4.3 Structure

The main sections of a Swagger definition are:

- definitions
  This is the class model for the service. It contains all the classes that may be used in the service operations.

- paths
  This section describes the functionality of the service. It contains all the paths for which a request can be created (except for the /api/swagger paths).

- x-ohibo-enumerations
  This section contains the enumerations referenced by scalar object members or parameters. Each enumeration contains a mapping between OHI internal values and external values.
  The OHI internal values are used in the OHI Back Office database. The external representations are used for interfacing with the client application.

- x-ohibo-modification-history
  This (optional) section lisitng modifications to the Swagger definition helps you locate new or modified operations and type definitions.

### 4.3.1 Paths and operations

The unit of work in a HSL service is the operation. An operation is a combination of a path and a HTTP verb (POST, GET, PUT, PATCH, DELETE)

Examples for the fictitious HSL_XYZ service:

- /xyz/v1/relation + POST
  Create a new relation.

- /xyz/v1/relation/{rel_nr} + GET
  Retrieve an existing relation.

- /xyz/v1/relation/{rel_nr} +PUT
  Replace an existing relation.

- /xyz/v1/relation/{rel_nr} +PATCH
  Selectively update an existing relation.

- /xyz/v1/relation/{rel_nr} +DELETE
  Delete an existing relation.

## 4.4 OHIBO-Specific extensions

The Swagger schema generated by the HSL application contains the following OHIBO-specific extensions:

- x-ohibo-revision
  The OHI revision number of the OHI internal schema definition. The revision number is used for tracing a specific HSL application to a change in the OHI source code control system.

- x-ohibo-enumerations
  List of enumerations containing:

  - domain : optional link to OHI BO domain

  - values: list of enumeration items. Each enumeration item has an external value and OHI internal value.

- x-ohibo-enum
  Associates a scalar value to an x-ohibo-enumeration.
  May apply to object members and parameters.

- x-ohibo-column
  Assigned to each scalar object member. Possible values:

  - 'none'
    The value of the object member is not (directly) related to an existing table.column value.

  - <table>.<column>
    The OHI table.column associated with this object member.

  - <empty>
    An empty value indicates an incomplete Swagger source specification.

### 4.4.1 Enumerations

Each enumeration item has an external value and an OHI internal value.

Example:

```
"RelationStatus" : {
     "values" : {
       "approved" : "A",
       "rejected" : "R"
     }
   },
```

As you can see, the external value ‘approved’ is mapped to an OHI internal value ‘A’. The external value is used for interfacing. The OHI internal value may be useful to understand OHI system messages triggered by a business rule violation or functional errors.

Note that when you use the PL/SQL interface you should use the external enumeration value when passing objects and parameters to the PL/SQL layer. Likewise, enumeration values in objects returned by the PL/SQL layer are automatically converted from OHI internal format to their external representation.

# 5 Configuration information

The following URI gives you information about a running HSL application:

https://<server>:<port>/<application>/dbinfo

For example

https://localhost:7002/HSL_POL/dbinfo

The following information is given:

- basePath
  Format: https://<server>:<port>/<application>/<context>
  This is the base URI for all operations in this service.

- database
  The name of the database associated with the current database connection.

- instance
  Instance name of the database associated with the current database connection.

- jndiName
  The JNDI name of the database connection (specified in the hsl.properties configuration file described in Doc[1])

- plsqlPackage
  The PL/SQL package which implements the operations of the HSL service.
  In this release, the revision number refers to the revision number of the code template used to generate the PL/SQL package. In a future release this will point to the revision number of the compiled PL/SQL package.

- user
  The database account used to log on to the database.

- user context
  The default OHI officer on whose behalf service operations are performed, as specified in the hsl.properties file.

Note that hsl.properties refers to the configuration file which is used to start the HSL service. The format of the hsl.properties file is described in **Doc[1]**.

# 6 Singular resource operations

A singular resource contains a single object.

### 6.1.1 Default members

By default, each singular resource has the following members:

- *links*
  a list of (HATEOAS) links to navigate to related operations on the same object (such as PUT, POST, PATCH, DELETE) or to navigate to nested objects or to re-request the object ('self').

- *id*
  An integer to uniquely identify the object. May not yet be uniformly implemented. See the Swagger definition or the online help in the OHI Back Office GUI application for more information.

## 6.2 GET

Return a singular resource.

### 6.2.1 Default filters:

- *expand*
  Possible values:

  - 'all'
    Include all nested objects

  - empty
    Do not include nested objects

  - <object.member>[,<object.member>]..
    Comma separated list to selectively include nested objects.

### 6.2.2 HTTP return codes

- 200: the resource is returned in the requested format.

- Other: see 'HTTP return codes'

## 6.3 PUT

Replace existing singular resource.
The inbound object must be complete.
Use PATCH for selective updates!

### 6.3.1 Default filters

Not applicable

### 6.3.2 HTTP return codes

- 200 (OK)
  A response object may be returned depending on the definition of this operation in the Swagger schema.

- Other: see 'HTTP return codes'.

## 6.4 POST

Create singular resource.
Normally a POST for a singular resource should be an operation on a collection resource.

### 6.4.1 Default filters

Not applicable.

### 6.4.2 HTTP return codes

- 201 (CREATED)
  A response object may be returned depending on the definition of this operation in the Swagger schema.

- Other: see 'HTTP return codes'

## 6.5 PATCH

Selectively update a singular resource.

### 6.5.1 Default filters

Not applicable.

### 6.5.2 HTTP return codes

- 200 (OK)
  A response object may be returned depending on the definition of this operation in the Swagger schema.

- Other: See 'HTTP return codes'

## 6.6 DELETE

Delete a singular resource.

### 6.6.1 Default Filters

Not applicable

### 6.6.2 HTTP return codes

- 200 (OK)
  A response object is not returned (because it has been deleted)

# 7 Collection resource operations

A collection resource consists of a list of singular resources.

For example, an AddressCollection object consists of a list of Address objects.

Normally the only verbs that apply on a collection resource are GET and POST.

### 7.1.1 Default members

By default, a collection resource has the following members:

- *items*
  A list of 0 or more singular resources.

- *links*
  A list of HATEOAS links to navigate to other operations on the collection resource or to its items.

- *totalResults*
  The number of results matching the search criteria. This is NOT the size of the items list.

- *limit*
  The number of items that may be returned by the GET operation.
  It is either:

  - the *limit* parameter which was passed to the request; or
  - the default *limit* value for this operation as described in the Swagger schema; or
  - the default *limit* value in the OHI Back Office tooling (10).

- *count*
  The number of items which were returned by the GET operation.
  This is a value between 0 and the value of *limit*

- *offset*
  Specifies the index of the first result to be returned (0 means that the first result is returned as the first item).
  Note that the ordering of the result set is determined by the implementation code. Beware that the contents of the results set may change between two invocations!
  The offset value is either:

  - the *offset* parameter which was passed to the request; or
  - the default *offset* value for this operation as described in the Swagger schema; or
  - the default *offset* value in the OHI Back Office tooling (0)

- *hasMore*
  Boolean indicating whether more results may be found with a subsequent call.
  True if *totalResults* > *offset* + *limit*

## 7.2 GET

Return a collection resource.

### 7.2.1 Default filters

- *Expand=value*
  Possible values:
  - 'all'
    Include all nested objects
  - empty
    Do not include nested objects
  - <object.member>[,<object.member>]..
    Comma separated list to selectively include nested objects.
- *limit=n*
  Limits the number of items to *n*.
- *offset=n*
  Indicates that the first item should be item #n of the search results.
  'offset=0' means that the list should start with the first search result.

### 7.2.2 HTTP return codes

- 200
  Return resource
- Other: see 'HTTP return codes'.

## 7.3 PUT

Not implemented.

## 7.4 POST

Create a new singular resource and add it to the collection.

### 7.4.1 Default filters

Not applicable.

### 7.4.2 HTTP return codes

- 201 (CREATED)
  The response body must contain the newly created resource.
  The Location response header should contain a URI to the newly created resource if the resource can be independently accessed.
  Form: 'Location: <uri>'

## 7.5 PATCH

Not implemented.

## 7.6 DELETE

Not implemented.

# 8 HATEOAS Links

HATEOAS links are server-provided links to help the REST client navigate through the server application.

See https://en.wikipedia.org/wiki/HATEOAS for more information about the use of HATEOAS links.

Both the PL/SQL interface and Java interface return HATEOAS links as part of a resource response object.

The HATEOAS link object as used in the HSL layer has the following members:

- *href*=<*absolute URI*>
  An absolute URI generated from an initial path after expanding templates, adding query parameters and adding the base path.

- *mediaType*
  Not currently used.

- *method*=<*get*|*put*|*post*|*patch*|*delete*>
  The HTTP verb associated with the link

- *profile*
  Not currently used.

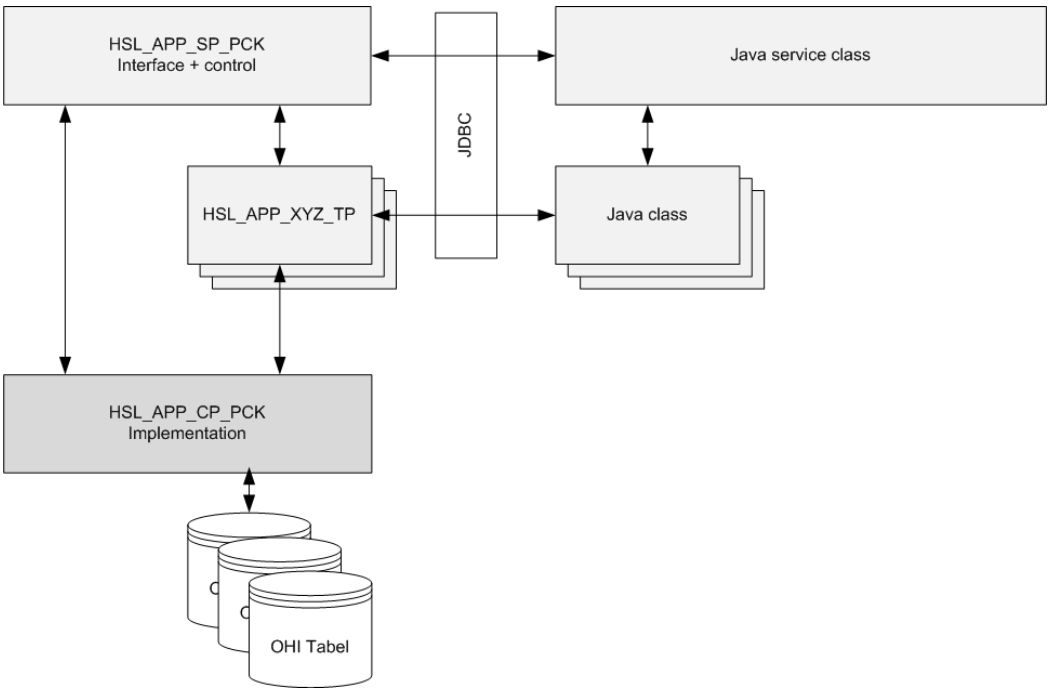
- *rel*=<*self*|*edit*>
  'self' is used to repeat the original request.

- *templated*=<*true*|*false*>
  A link is templated if it is generated from a templated path. An example of a templated path is '/policy/{number}/member'.

# 9 Invocation from PL/SQL

HSL services can be accessed through the RESTful HTTP interface and through PL/SQL. This means it is possible to create custom applications which access HSL services through PL/SQL. This chapter provides a few pointers for interfacing with HSL services through PL/SQL.

In the diagram below, HSL_APP_SP_PCK indicates the PL/SQL package which is the interface to the Java service layer and which can be invoked from PL/SQL:




The following fragment illustrates how a HSL service operation can be invoked through PL/SQL:

```
declare
  l_call_context hsl_cmn_call_context_tp := hsl_cmn_call_context_tp();
  l_return_context hsl_cmn_return_context_tp;
  l_output hsl_pol_simple_policy_coll_tp;
begin
  alg_trace_pck.enable;
  l_call_context.m_language := hsl_cmn_string4000_tp('nl-NL'); -- language
  l_call_context.m_user_context := hsl_cmn_string4000_tp('MANAGER'); -- usercontext

  hsl_pol_sp_pck.simplepoliciesbycco
  ( pi_call_context   => l_call_context
  , po_return_context => l_return_context
  , po_output => l_output
  , pi_expand => 'all'
  , pi_limit => 10
  , pi_collectiveagreementnumber => 329
  , pi_referencedate => null
  , pi_offset => 0
  );

  l_return_context.print('l_return_context'); -- print HTTP return code
  l_output.print('l_output'); -- print collection resource
end;
/
```

Notes:

- The call to `alg_trace_pck.enable` only serves to enable dbms_output. It should not be used in production mode, because:
    - The use of alg_trace_pck.enable prevents the reset of the package state.
    - The alg_trace_pck package is not normally granted to the HSL user
- The call context (see 'SQL Types' later in this chapter) contains generic meta data to control the transaction.
- The user context must refer to a valid OHI BO officer who will be associated with the transaction.
- The language of the call context can be used to override the language of the OHI BO officer selected with the 'user context'.
- Pagination is handled at the PL/SQL level and controlled through the pi_limit and pi_offset parameters.
- The 'expand' parameter controls which sub-objects are included. A value of 'all' means that all sub-objects are included.

## 9.1 Oracle Account

You need a database account to call the plsql implementation. This must NOT be the OHI object owner or you will run into HTTP 412 (object owner is not allowed to execute HSL operations).

**Doc[1]** describes how to set up an account for using HSL services.

Note that the OHI object owner will need to grant access to the HSL user account. See the example below how it is done for the fictitious account *hsl_test*:

```
begin
  alg_security_pck.hsl_grants(pi_grantee=> 'HSL_TEST');
end;
/
```

## 9.2 Which Package?

Each HSL service is associated with a single interface package with procedures and functions that can be invoked by custom applications.

The name of the interface package can be derived from the WAR file associated with the service.

Given a service XYZ, its WAR file would be HSL_XYZ.war.The corresponding PL/SQL interface package would then be HSL_XYZ_SP_PCK.

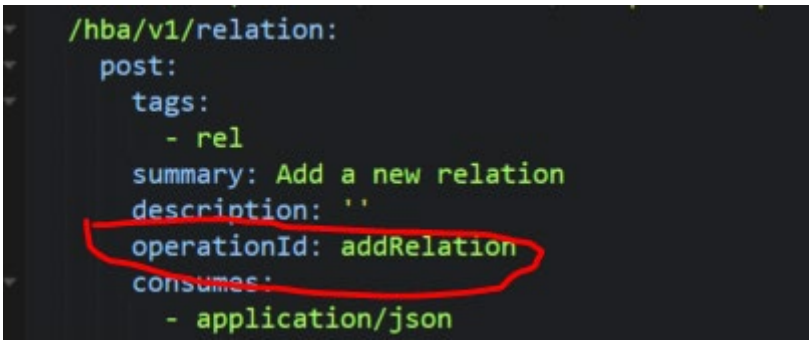
So for the POL service, the interface package is HSL_POL_SP_PCK.

## 9.3 Mapping Operations to Packaged Procedures

When using the HTTP interface, each operation is a combination of a path and a HTTP verb.

In the Swagger definition, each operation is given a unique name, called 'operation ID'.

In the example below the operation ID for /hba/v1/relation + POST is defined as 'addRelation':




From the application name 'HBA' we can deduce that the interface package is called 'HSL_HBA_SP_PCK'.
The procedure name is mapped to the operationId.
So if the application name is 'HBA' and the operation ID is 'addRelation', the corresponding PL/SQL packaged procedure is HSL_HBA_SP_PCK.addrelation.

## 9.4 Invoking an Operation

The interface package provides a packaged procedure for every service operation.

The interface of every operation procedure is similar.

```
procedure dosomething
( pi_call_context   in    hsl_cmn_call_context_tp
, po_return_context   out hsl_cmn_return_context_tp
, pi_some_inbound_param
, po_output
);
```

Each procedure has the following parameters:

- pi_call_context
  Inbound object containing runtime metadata such as the base path, language, and OHI officer.

- po_return_context
  Outbound object containing HTTP code, and technical and functional message if an error occurred.

Our first example is a procedure to add a relation:

```
procedure addrelation
( pi_call_context   in    hsl_cmn_call_context_tp
, po_return_context   out hsl_cmn_return_context_tp
, pi_relation in hsl_hba_relation_tp
, pi_forceupdate in varchar2 default 'true'
, pi_expand in varchar2
);
```

This procedure has several inbound parameters including an inbound object pi_relation. It has no outbound object, meaning that the calling application will get a HTTP code but no content.

Our second example is a procedure to patch a relation:

```
procedure patchrelation
( pi_call_context   in    hsl_cmn_call_context_tp
, po_return_context   out hsl_cmn_return_context_tp
, po_output out hsl_hba_relation_tp
, pi_relation in hsl_hba_relation_tp
, pi_expand in varchar2
);
```

This operation was designed to return the updated copy of the object.

Note that all operations may process at most one inbound object and return at most one outbound object.

## 9.5 SQL types

Whereas Java classes are the containers to hold objects in Java, SQL types are the containers to hold objects in PL/SQL.

### 9.5.1 Service-specific types and common types

HSL services have their own SQL types which are not shared with for example SVL services.

Most HSL types are not even shared with other HSL services.They are prefixed with HSL_<APP>.

Example: all complex SQL types for the REL service are prefixed HSL_REL

Common types may be used by multiple HSL services. They are prefixed HSL_CMN designating their common use.
Common types are used for scalar values or generic metadata. They are shared because the definitions of these types are unlikely to change over time.

Examples of service-specific types:

- hsl_rel_preferred_acco_tp
  type definition to hold preferred account details for the REL service.
- hsl_rel_link_tp
  type definition to hold link details for the REL service
- hsl_pol_policy_tp
  type definition to hold policy data for the POL service.
- hsl_pol_link_tp
  type definition to hold link details for the POL service.

Examples of common types:

- hsl_cmn_call_context_tp
  generic definition to hold call context of an operation.
- hsl_cmn_return_context_tp
  generic definition of a return context holding HTTP result code and error messages

- hsl_cmn_string30_tp
  holds a single varchar2(30) value

- hsl_cmn_date_tp
  holds a single date value.

### 9.5.2 Call Context

The call context object is passed to every operation to pass metadata:

It contains the following information:

- m_base_path
  The basepath is the URI which is prepended to create the absolute links needed by the calling application.

- m_caller_id
  The caller ID may be set to identify the end user on whose behalf the HTTP request was generated. For example the relation number of an insurance member, care provider or broker.
  See the documentation of the HSL service operation if the m_caller_id must be set.

- m_caller_role
  The caller role may be set to indicate the role of the end user on whose behalf the HTTP request was created.
  Theoretically it is possible that a care provider is also an insurance member. A request to retrieve claims would then be ambiguous: is the caller an insurance member wishing to retrieve his own claims or a care provider wishing to retrieve claims associated with his services?
  Together, the caller ID and caller role should provide an unambiguous context to the operation.
  See the documentation of the HSL service operation if the m_caller_role must be set.

- m_language (optional)
  End user language, for example nl-NL
  See 'Accept-Language' in 'Creating HTTP requests' earlier in this document.

- m_user_context (mandatory)
  Indicates the OHI officer on whose behalf the request is processed. Must be an existing Oracle database account name and refer to an existing and time valid row in ALG_FUNCTIONARISSEN.
  The OHI officer may be an employee of the OHI customer, functional administrator, call center employee, or an account registered for self-service actions.

### 9.5.3 Return context

The PL/SQL implementation of each operation passes a return context to the caller. The caller is either the service class in the Java layer providing the HTTP RESTful interface or PL/SQL custom code.

The return context is a SQL type with the following members:

- m_code
  a HTTP return code, eg. 200 (OK)

- m_functional_message
  May be set if an exception occurred while processing the operation.

- m_technical_message
  May be set if an exception occurred while processing the operation.

### 9.5.4 Built-in functionality of HSL types

Each service-specific SQL type in a HSL service (example: HSL_POL_POLICY_TP) has the following generic functionality:

- constructor
  The constructor creates a new object and sets appropriate default values for scalar values if required by the original OHI Swagger definition.

- example
  This function populates an object with example data, derived from the original OHI Swagger definition.

- o2x
  Creates an external representation of an object to be handed to the calling application.

- x2o
  Creates an internal representation of an object (which is used by the OHI implementation code).

- print
  prints the content of an object and its nested objects using dbms_output.put_line

- scalars_to_str
  returns a varchar2 string formatted as 'name1="value1",name2="value2"' etc. Used by OHI implementation code.

- class_name
  Returns the name of the Java class corresponding witht this SQL type.

## 9.6 GET Example

The following example calls a GET-operation to retrieve a single resource:

```
declare
  l_call_context hsl_cmn_call_context_tp := hsl_cmn_call_context_tp();
  l_return_context hsl_cmn_return_context_tp;
  l_output hsl_pol_policy_tp;
begin
  alg_trace_pck.enable;
  l_call_context.m_base_path := hsl_cmn_string4000_tp('http://ol6ohi:4321/HSL_pol');
  l_call_context.m_language := hsl_cmn_string4000_tp('nl-NL');
  l_call_context.m_user_context := hsl_cmn_string4000_tp('MANAGER');

  hsl_pol_sp_pck.policybynumber
  ( pi_call_context   => l_call_context
```

```
  , po_return_context => l_return_context
  , po_output => l_output
  , pi_expand => 'all'
  , pi_number => 1764
  , pi_referencedate => null
  );

  l_return_context.print('l_return_context');
  l_output.print('l_output');
end;
                      /
```

# 10 Language Aspects

The session language in the PL/SQL session (which implements the HSL service operation determines) the representation of:

- Messages
- Language dependent data (example: multilingual product description)

Selecting the correct session language has an impact on the functioning of the HSL application!

The default language in the PL/SQL session is the preferred language of the OHI officer which is set through the call context. If you are using the Java interface, the call context is set through the hsl.properties configuration file set up by your OHI technical administrator (see **Doc[1]**).

The session language can be overruled using the HTTP request header 'Accept-Language' as described in 'Generic Usage Aspects of HSL Services'.

Note that the session language is set for each request.

This language functionality might be used to address a customer in a customer specific language.

## 10.1 Current Behaviour

Setting the HTTP request header 'Accept-Language' will set the language for the session.

### 10.1.1 Messages

Messages will be given in the session language. This behaviour is in line with the GUI application.

### 10.1.2 Language-dependent data

Language-dependent data will be displayed and processed in the session language. This behaviour is in line with the GUI application.

### 10.1.3 Known Issue: HSL_POL service

If set, the Back Office parameter value for 'Polis use case service > functionaris' will override the OHI officer.
In that case, the session language will be overruled by the preferred language of the OHI officer associated with the Back office parameter value for 'Polis use case service > functionaris'.

### 10.1.4 Known Issue: HSL_REL service

If set, the Back Office parameter value for 'Relatie use case service > functionaris' will override the OHI officer.
In that case, the session language will be overruled by the preferred language of the

OHI officer associated with the Back office parameter value for ‘Relatie use case service > functionaris’.

# 11 Terminology

| Term | Meaning | Example |
|---|---|---|
| **Application** | Unit of deployment for a HSL service | |
| **Attribute** | A scalar member of a resource | |
| **Basic Authentication** | a method for an HTTP user agent to provide a user name and password when making a request. | |
| **Body Parameter** | A resource passed as the payload of a request | |
| **Collection Resource** | A resource consisting of metadata attributes and a list of singular resources | |
| **Deserialization** | Conversion of a character string (JSON format) to an object. | |
| **Enumeration** | a set of allowed (string) values | |
| **HATEOAS** | A follow-up link returned as part of the response to a REST operation to help the client operation navigate to an appropriate next request. | |
| **Header** | a name/value pair in a HTTP request or response. | Accept-Language:nl-NL |
| **HSL** | HTTP Service Layer - the technical implementation for OHI Back Office Use Case services. | |
| **HTTP Code** | Standardized return code for a HTTP request. | 200 (OK), 201 (CREATED), 400 (BAD_REQUEST), 404 NOT_FOUND), 405 (METHOD_NOT_ALLOWED), 500 (INTERNAL_SERVER_ERROR) |
| **HTTP Verb** | A value from the following set: GET, PUT, PATCH, POST or DELETE. The verb and URI together define the required service operation. | |
| **JSON** | A standard format for serializing objects to ASCII strings (and vice versa) | |
| **Object** | An object in the object model of a REST service. Same as 'type'. | |
| **Operation** | A single action on a resource in a RESTful service | |
| **Pagination** | Creation of a subset when | |

| | | |
|---|---|---|
| | returning a collection resource. | |
| **Parameter** | A parameter to a HTTP request. See also 'Body Parameter', 'Query Parameter', and 'Path Parameter' | |
| **Path Parameter** | A parameter which is part of the path. | /pol/v1/simplePolicies/123 |
| **Query Parameter** | A parameter which is added to the path. | /pol/v1/simplePolicies/123?referenceDate=2011-12-31 |
| **Resource** | An object which is passed to, or returned by a service operation. | |
| **RESTful Service** | A HTTP-based web service following the REST application architecture. | |
| **Return code** | HTTP Code returned by a service operation. | |
| **Serialization** | Conversion of an object to a machine-independent format. | |
| **Service** | A group of operations and its object model. The service interface is defined by the 'Swagger Schema'. The service is deployed as a (WAR) application. | |
| **Singular Resource** | A single object which is passed to, or returned by a service operation. See also 'Collective Resource'. | |
| **Swagger Schema** | The specification document that describes the interface of a RESTful service. | |
| **Tag** | A logical category for grouping operations in a Swagger schema. An operation may have multiple tags. | |
| **Templated Path** | A path with one or more path parameters. | |
| **Type** | An object in the object model of a REST service. Same as 'object' | |
| **WAR** | File format for deploying web applications such as HSL services. | |

# 12 Appendix A – HSL C2B services – Usage points of attention

Starting with OHI Back Office release 10.17.2.2.0, a set of web service operations has been made available in the HTTP Service Layer in order to replace the 'old' Connect to Back Office (C2B) services that modified policies and relations. These SOAP envelope based services were first delivered in 2006 and have been used for a long time by many OHI customers to send all kinds of policy and relation modification requests to OHI Back Office.

The new C2B web service operations in HSL offer – as far as is possikle - identical functionality as was offered by the 'old' C2B. An explanation for this is given in a separate Appendix of the HTTP Service Layer installation manual.

Because of the different technology used, the new service operations may react differently from the old ones. The most important usage aspects are discussed now.

## 12.1 Transaction handling

Each C2B web service call fails or commits atomically. No partial change will ever be committed. However, transaction handling is done at the database level. When the network fails between application and database server the response message may not be received but the transaction may have succeeded. This is analogous to the 'old' C2B implementation. In the asynchronous version of the 'old' C2B, the response queue might not be filled while the transaction was successfully processed. No 'two-phase commit' functionality was offered for the dequeue and service call, because such distributed transaction handling is complicated and heavyweight, and risky. It would require complex administrative corrections when such transactions fail. For that reason supporting idempotence would be a better option. This idempotence was not present in the old implementation and is not present in the new implementation either.

Because the transaction is processed within the OHI Back Office database, consistent and isolated processing is already guaranteed, due to the way the Oracle database offers transaction isolation and due to the way all business rules within OHI Back Office are implemented, using an appropriate level of locking where necessary.

## 12.2 Idempotence

The service operations of C2B do not offer idempotence behaviour by default, in order to act identically as they did when originally created. The result of 2 subsequent identical calls may differ per operation. This historical functionality may lead to unwanted results in combination with the implemented transaction handling but has not caused serious business problems over the past years.